1992-12

Implementation of cyclic spectral analysis methods

Carter, Nancy J.

Monterey, California. Naval Postgraduate School

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): EC	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

IMPLEMENTATION OF CYCLIC SPECTRAL ANALYSIS METHODS (U)

12. PERSONAL AUTHOR(S)
Carter, Nancy J.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1992	112

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	cyclic spectral analysis, cyclostationary, cyclic spectrum, strip spectral correlation algorithm, fast fourier transform accumulation method

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Phase-shift keyed (PSK) signal modulation methods are coming into increasing use in modern communications. This thesis describes the implementation of three methods of computing the cyclic spectrum to determine the presence of PSK signals. The Strip Spectral Correlation Algorithm (SSCA) and the Fast Fourier Transform (FFT) Accumulation Method (FAM) both estimate the cyclic spectral plane. The Sub-FFT Accumulation Method (SUBFAM) program computes the Spectral Correlation Function (SCF) for a set of possible spectral frequencies for a single cycle frequency. The results of algorithm performance are presented along with a discussion of promising areas for performance enhancement and automation of signal detection and classification.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Herschel H. Loomis, Jr.	22b. TELEPHONE (Include Area Code): (408) 656-3214	22c. OFFICE SYMBOL: EC/LM

Approved for public release; distribution is unlimited.

Implementation of Cyclic Spectral Analysis Methods

by

Nancy J. Carter
Lieutenant Commander, United States Navy
B.S., University of Maryland College Park, 1981

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

ABSTRACT

Phase-shift keyed (PSK) signal modulation methods are coming into increasing use in modern communications. This thesis describes the implementation of three methods of computing the cyclic spectrum to determine the presence of PSK signals. The Strip Spectral Correlation Algorithm (SSCA) and the Fast Fourier Transform (FFT) Accumulation Method (FAM) both estimate the cyclic spectral plane. The Sub-FFT Accumulation Method (SUBFAM) program computes the Spectral Correlation Function (SCF) for a set of possible spectral frequencies for a single cycle frequency. The results of algorithm performance are presented along with a discussion of promising areas for performance enhancement and automation of signal detection and classification.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The field of cyclic spectral analysis is growing in importance as non-stationary modulation schemes proliferate in modern communications. Traditionally, spectral analysis methods have assumed that signals of interest were stationary or had non-time-varying statistical parameters. This assumption is untrue for the cyclostationary signals created when using pulse or carrier amplitude modulation, phase or frequency carrier modulation or digital pulse or carrier modulation. Gardner [Ref. 1:pp 419-453] discusses many of these signal types. Spread spectrum modulation techniques such as direct sequence PSK and frequency-hopped FSK also exhibit temporally varying spectral features [Ref. 1:pp 453-457]. Reference 2 provides a tutorial on cyclic spectral analysis.

Cyclic spectral analysis methods take advantage of spectral variation over time by correlating spectral estimations at discrete time intervals to locate signals that may not otherwise be identified. A direct sequence signal with a very wide bandwidth may be indistinguishable from a noisy background in the frequency spectral estimate of Figure 1 [Ref. 3:pp 2-9 - 2-10]. But this signal becomes

clearly identifiable in the cyclic spectral estimate of Figure 2.

One cyclic spectral technique is the frequency smoothed cyclic periodogram method (FSM) [Ref. 1:pg 464]. This has been implemented in the Cyclic Spectral Analysis Software Package [Ref. 4] as the program *sxaf*. Hereafter, this software will be referred to as the SSPI package or the FSM. This method was used to generate Figures 1 and 2. Unfortunately, the FSM is less efficient than time smoothing methods for general spectral estimation [Ref. 5:pg 38].

B. THESIS GOALS

The purpose of this thesis is to implement two time smoothing algorithms, the FFT Accumulation Method (FAM) [Ref. 5:pp 44-47] and the Strip Spectral Correlation Algorithm (SSCA) [Ref. 5:pp 47-48]. The FAM is less computationally complex than the FSM and the SSCA less so than the FAM. Both methods have been written to be compatible with the SSPI package. By integrating FAM and SSCA into the SSPI package future research may more easily be done in algorithm performance comparison and enhancement.

Both the FAM and SSCA accept input data generated by the SSPI utility program *pam*. They also generate results compatible with the SSPI plotting utility *plot_sxaf*. Each program may optionally generate output in a format compatible with the MATLAB [Ref. 6] package which is in common use at the

Naval Postgraduate School. Input and output data are in ASCII file formats.

Throughout this thesis the same BPSK signal data is used to illustrate the implementations of FAM and SSCA. A representative signal sample is plotted in Figure 3. No noise was added to this signal. By estimating the frequency spectrum at different points in time, it is evident in Figure 4 that the spectral features do indeed vary temporally.

Chapter II describes the implementation of the FAM. Chapter III describes the implementation of the SSCA. Chapter IV describes a utility to estimate cycle frequencies for a given baseband frequency. The Sub-FFT Accumulation Method (SUBFAM) program [Ref. 7] gives a quick result for a specific subset of the spectral plane. Chapter V discusses specific algorithm performance comparisons. Finally, Chapter VI presents conclusions and recommendations for future areas of research.

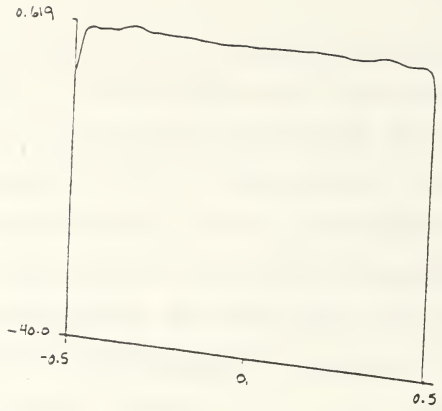

Figure 1 Power Spectral Density of BPSK Signal in Noise [Ref. 4:pg 18]

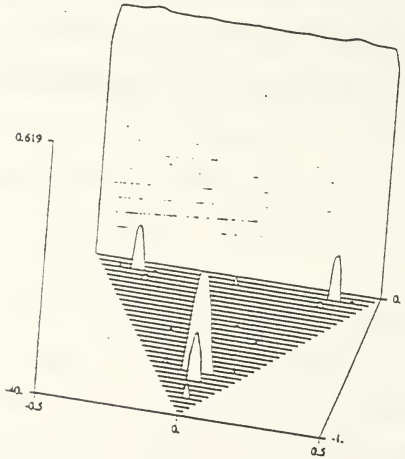

Figure 2 Cyclic Spectral Estimate of Direct Sequence Signal in Noise [Ref. 4:pg 18]

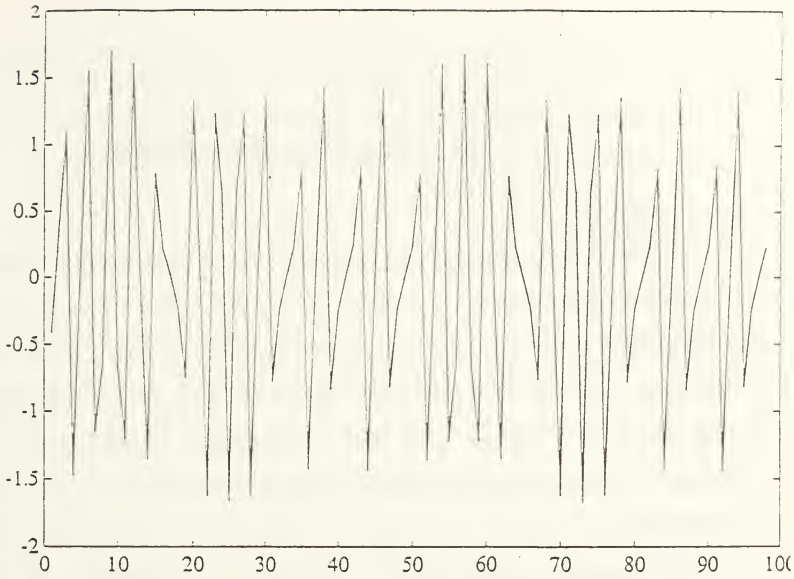

Figure 3 Typical BPSK Signal Data

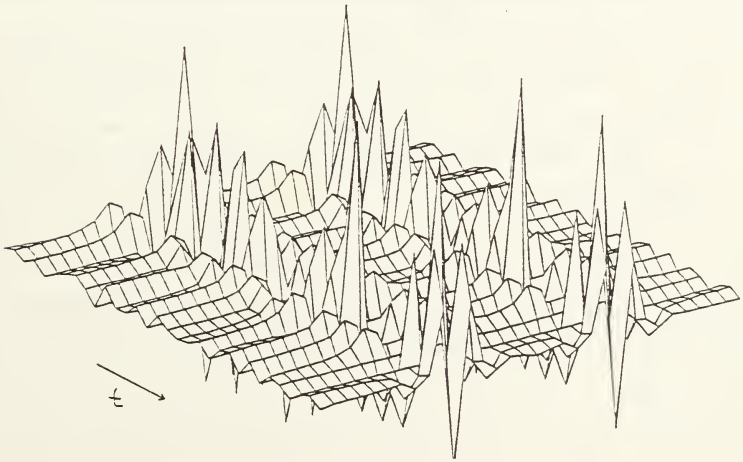

Figure 4 Time Sequence Spectrum of Typical BPSK Signal

II. FFT ACCUMULATION METHOD

A. THEORY

The FFT Accumulation Method (FAM) [Ref. 5:pp 44-47] is a Fourier transform of correlation products between spectral components smoothed over time. Periodicities in the spectral components then become detectable. The cyclic spectral plane as shown in Figure 5 ranges in frequency from $-f_s/2$ to $+f_s/2$, and in cycle frequency from $-f_s$ to $+f_s$ where f_s is the sampling frequency. For each unique (f_j, α_q) cell in Figure 5, the FAM cross-spectral estimate is defined to be [Ref. 5:pg 44]:

$$S_{XY_T}^{\alpha_o+q\Delta\alpha}(nL, f_j)_{\Delta t} = \sum_{r=0}^{P-1} X_T(rL, f_k)\, Y_T^*(rL, f_\ell)\, g_c(n-r)\, e^{\frac{-i2\pi rq}{P}} \qquad (1)$$

The FAM auto-spectral estimate is defined to be [Ref. 5:pg 44]:

$$S_{XX_T}^{\alpha_o+q\Delta\alpha}(nL, f_j)_{\Delta t} = \sum_{r=0}^{P-1} X_T(rL, f_k)\, X_T^*(rL, f_\ell)\, g_c(n-r)\, e^{\frac{-i2\pi rq}{P}} \qquad (2)$$

where: j is the average frequency bin $(k+\ell)/2$

q is the difference or bandwidth frequency $(k-\ell)$

$g_c(n-r)$ represents the Hamming window.

The FAM was implemented by forming an array from x(kT) $(0 \leq k \leq N-1)$ with rows which are N' points long from the input sample data. The starting point of each succeeding row is offset from the previous rows starting position by L samples. A Hamming window is applied across each row which is then Fast Fourier transformed and downconverted to baseband. The result at this point is a two-dimensional array with columns representing constant frequencies. Each column was point-wise multiplied in turn with the conjugate of the columns resulting from processing y(kT) $(0 \leq k \leq N-1)$. Each resultant product vector was Fast Fourier transformed and the low frequency half placed into the final cyclic spectral plane at the appropriate location in the cyclic spectral plane. Figure 6 shows a block diagram for the FAM cross-spectral estimate.

The following subsections in this chapter discuss in detail what has been described in the previous paragraph. The cycle frequency resolution of FAM is $\Delta\alpha = f_s/PL$ [Ref. 5:pg 44] where f_s is the original data sampling rate. The frequency resolution of FAM is $\Delta f = f_s/N'$ [Ref. 5:pg 44]. The time-frequency resolution product is $\Delta t\Delta f = PL/N'$ [Ref. 5:pg 45].

The command line format for calling the FAM program is provided in Appendix A. The FAM source code listing is provided in Appendix B.

B. IMPLEMENTATION

1. Input Channelization

The input sample data is formed into a two-dimensional array. The array row length is equal to the number of input channels N'. For a given number of input sample points N, a row size of N', and a chosen offset L, there are P = (N-N')/L ≈ N/L rows formed. The choice of N' must take into consideration that ideally the time-frequency resolution product must be much greater than one [Ref. 5:pg 40]. N' should also be a power of 2 to avoid truncation or zero-padding in the FFT routines. L should be chosen to be less than or equal to $N'/4$.

The completely filled array is P rows by N' columns. Figure 7 shows how a small array is filled from a discretely sampled signal x(kT) when N'=16, P=8 and L=4. The number inside each cell represents the value of k used to index on x(kT) to fill that location in the array.

Figure 8 shows the magnitude of the original data for the example BPSK signal organized into the P by N' array where N=512, N'=32, L=4 and P=128. The input data is assumed to be complex with a real and imaginary component to each sample

point. Figure 9 shows a single row of the array. The phase changes of the BPSK are evident.

2. Windowing

A Hamming window [Ref 7:pg 467] is applied to each row of the array. The equation for the Hamming window is:

$$w(r) = 0.54 - 0.46\cos\left(\frac{2\pi r}{N'-1}\right), \quad 0 \le r \le N'-1 \qquad (3)$$

A 32-point Hamming window is plotted in Figure 10. It is applied to both the real and imaginary parts of the complex example array. The magnitude of the resultant array is shown in Figure 11.

3. First FFT

Each row of the windowed data array is Fast Fourier transformed to reveal the first spectral components. The resultant array is still indexed P rows by N' columns but now the column index relates to a specific bin of spectral frequencies. Figure 12 illustrates this relationship. Figure 13 shows the results of FFTing the BPSK example.

4. Downconversion

Each row of spectral components is downconverted to baseband through multiplication with the complex exponential,

$$e^{\frac{-i2\pi kmL}{N'}}$$

where: m is the row index, $0 \leq m \leq P-1$

k is the column index, $0 \leq k \leq N'-1$

The magnitude of the exponential is unity over the array but the phase shows considerable variation. Figure 14 shows the phase of the exponential over the P by N' array. Figure 15 shows the phase for one representative row. The magnitude of the array remains unchanged from Figure 13.

5. Multiplication

For each cell in the cyclic spectral plane, one column of the array is multiplied with the conjugate of another column. The separation of the columns is determined by the desired frequency separation or cycle frequency ($\alpha=f_k-f_\ell$). The mid-point between the columns is the frequency bin of interest ($f=(f_k+f_\ell)/2$). Figure 16 shows two columns to be conjugate multiplied for use in filling a specific f,α cell. Figure 17 illustrates the f,α bin values in the cyclic spectral plane for $N'=4$. Figure 18 shows the corresponding two column indices used to form the product vector which is used to produce the cells of Figure 17.

6. Second FFT

The product from the previous multiplication is FFT'd to yield a P-point result. Only the middle of the resulting spectrum is retained and stored into the designated f,α cell. The upper and lower ends are undesireable because of increased

estimate variation at the channel pair ends [Ref. 5:pg 45]. Figure 19 illustrates which part of the estimate is retained and placed into the cell. Figure 20 shows the final result of the FAM auto-spectral process on the example BPSK signal.

7. Data Reduction

The FAM program typically generates large output data files. For convenience, an option may be chosen to reduce the amount of output. By comparison sorting for the largest α value in an f,α cell, the number of α slices output is reduced from $N'P/2$ to N'. Overall FAM program execution time increases accordingly to accomplish the sorts.

Figure 21 illustrates the output full cyclic spectral plane storage array before sorting. Figure 22 shows the output array after data reduction has been completed. Figures 23 and 24 plot the resulting spectral half-planes without and with data reduction respectively.

Since all cells have an equal number of α values, all sorts are of equal complexity. Further work on data reduction would require selection of a largest α value from widely varying numbers of α values depending on the choice of N, N' and L.

C. PERFORMANCE

An established method of evaluating the complexity of an algorithm is to determine the number of floating point

operations that must be performed. For this estimate it is assumed that an auto-spectral estimate is being computed and the data is complex in every step. It is also assumed that the Hamming window coefficients and the downconversion coefficients have been previously computed and stored for later use. Each N-point FFT requires $(N/2)*\log_2 N$ complex floating point multiplies [Ref. 8:pg 506] or $4*(N/2)*\log_2 N$ real floating point multiplies. The cost of any output data reduction is not considered here.

Applying the window.....$2*P*N'$

First FFT.....$2*P*N'*\log_2 N'$

Downconversion.....$4*P*N'$

Multiplication.....$4*N'*N'*P$

Second FFT.....$2*N'*N'*P*\log_2 P$

Total: $(6+4*N')*P*N' + (2*P*N')*(\log_2 N' + N'*\log_2 P)$ (4)

Since $P \approx N/L$ (5)

Total: $(6+4*N')*NN'/L + (2*NN'/L)(\log_2 N' + N'\log_2 N/L)$ (6)

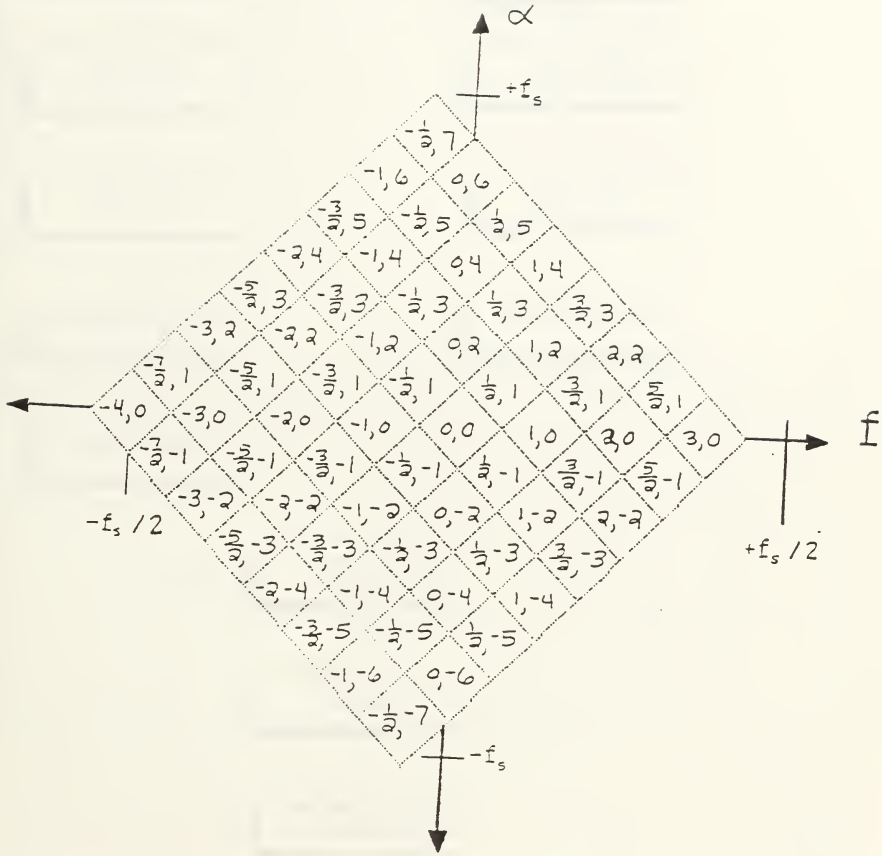

Figure 5 Generic FAM Cyclic Spectral Plane

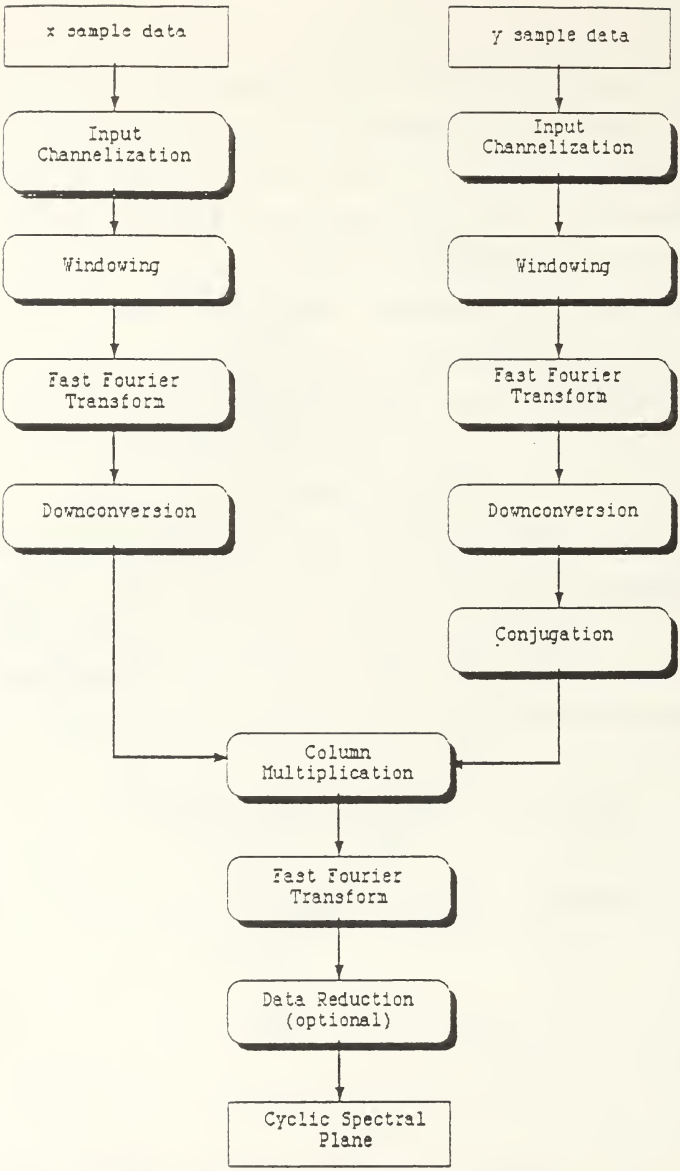

Figure 6 Block Diagram of FAM Cross-spectral Estimate

	Column Index															N'-1=
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
3	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
4	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
5	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
6	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
P-1=7	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

Figure 7 Layout of A Sample Input Array Showing Input Sample Storage for N'=16, N=48, L=4 and P=8.

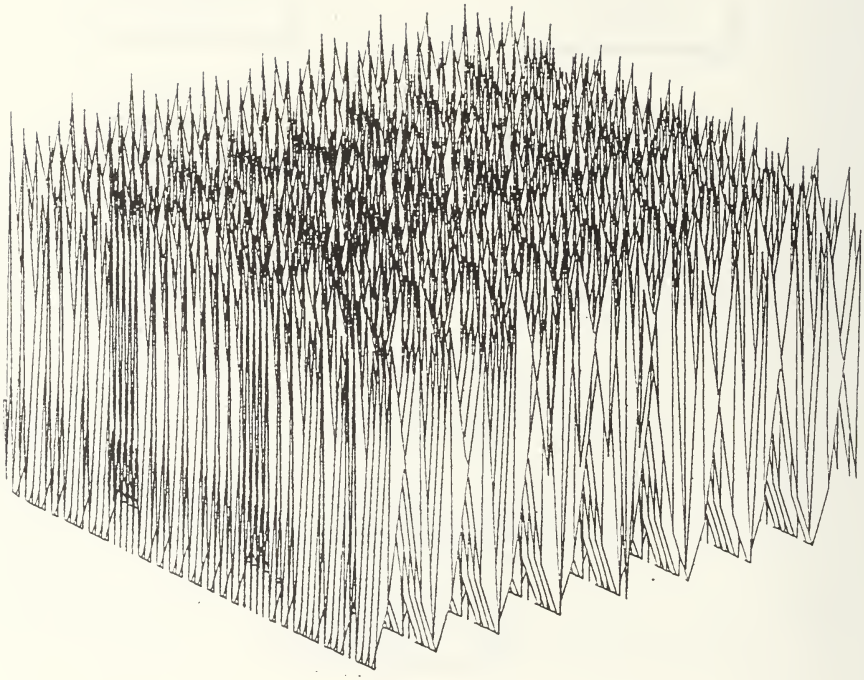

Figure 8 Example BPSK Signal Data Array, N'=32, P=128, L=4

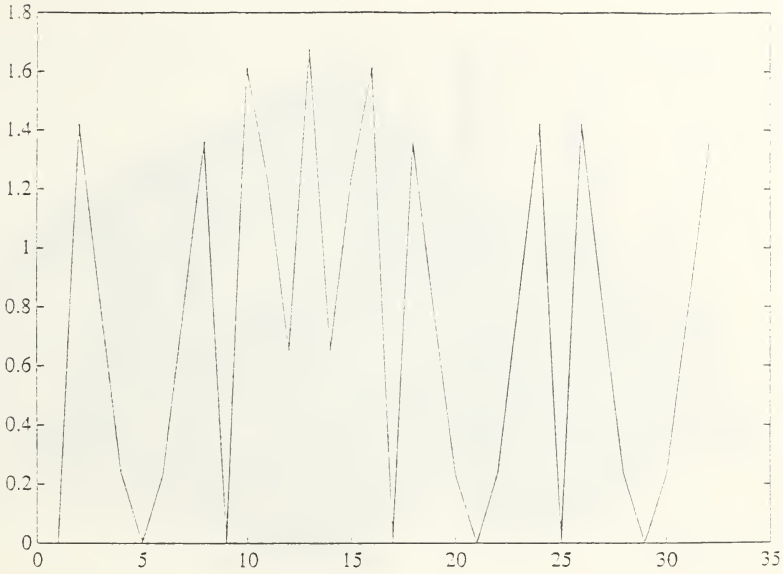

Figure 9 Single Row of BPSK Signal Input Array

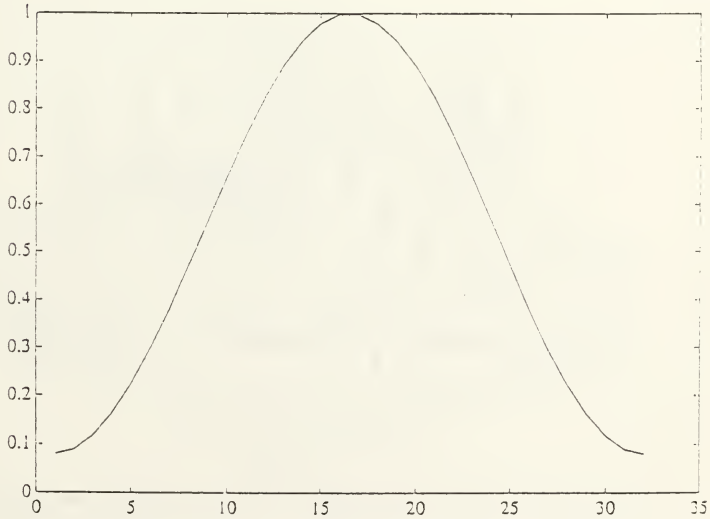

Figure 10 Thirty-two Point Hamming Window

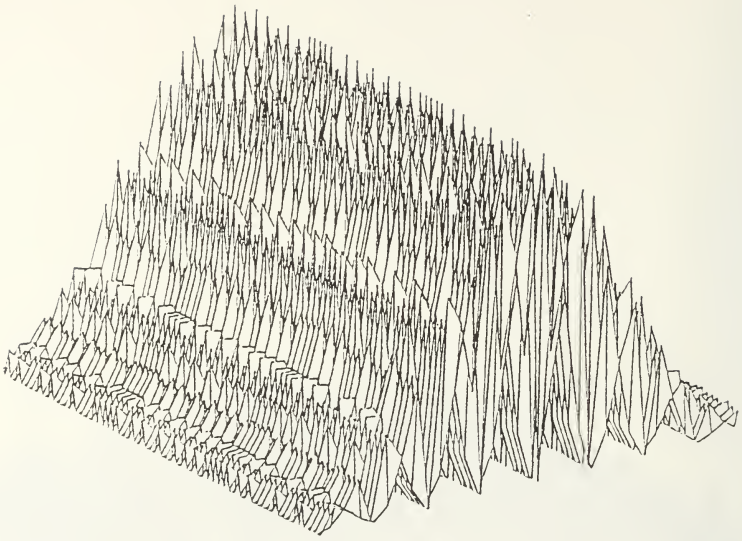

Figure 11 Example BPSK Signal Data After Windowing

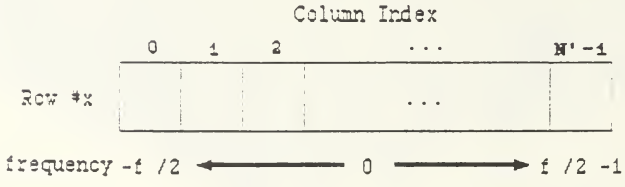

Figure 12 Generic Array Row After First FFT

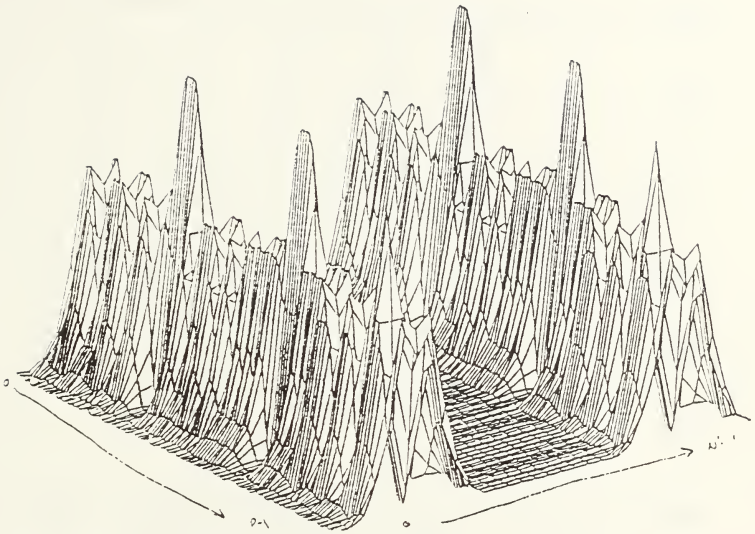

Figure 13 Example BPSK Signal Data After First FFT

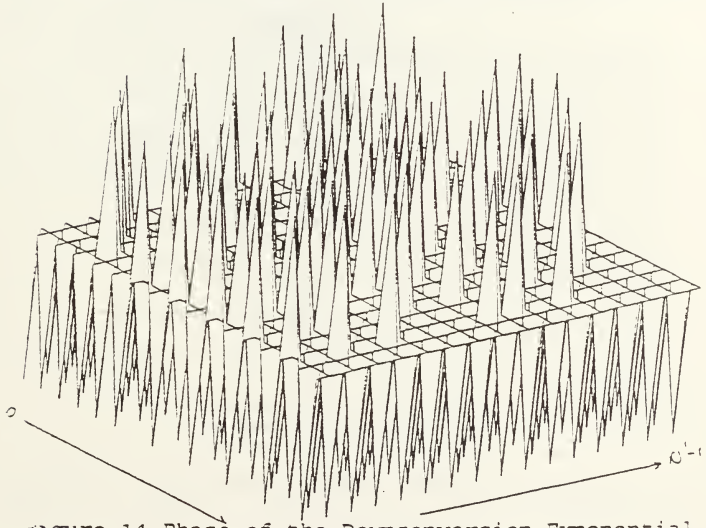

Figure 14 Phase of the Downconversion Exponential

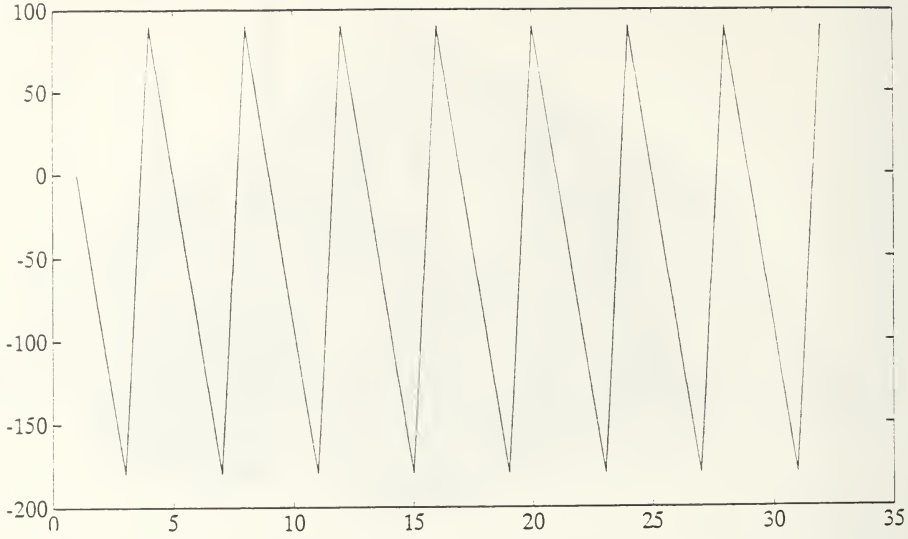

Figure 15 Phase For a Single Row of Array

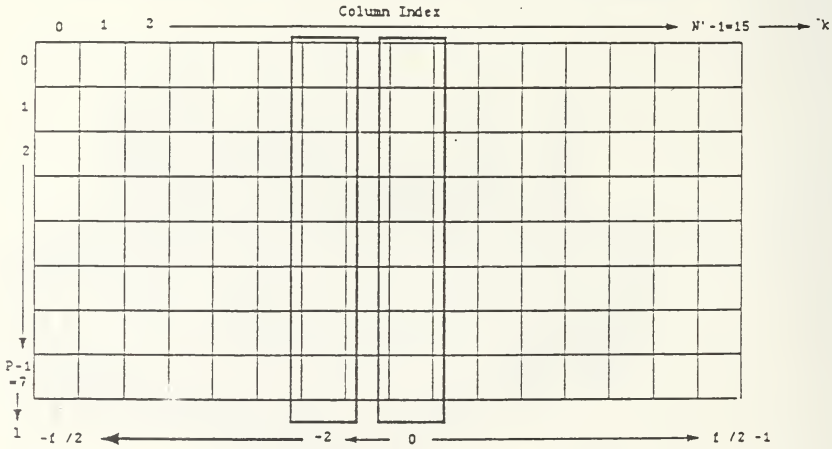

Figure 16 Generic Array Showing Two Columns to be Multiplied, f = -1, α = 2

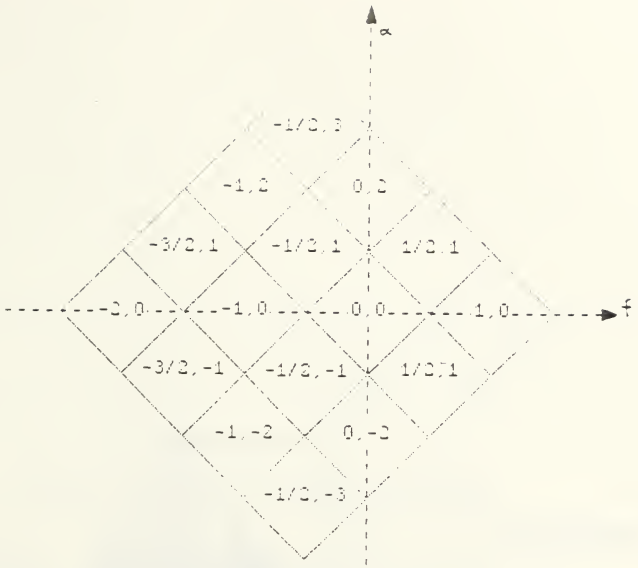

Figure 17 Generic Array Showing f,α values

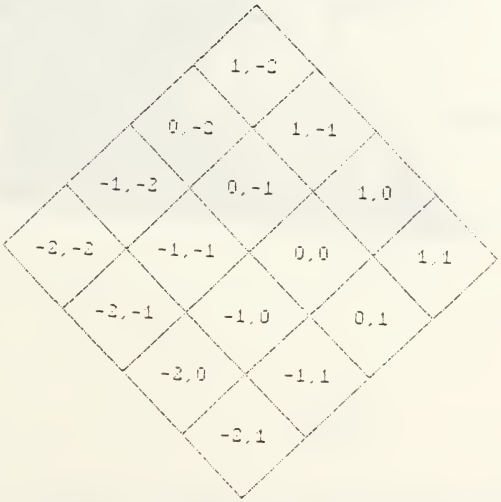

Figure 18 Generic Array Showing Column Index Values

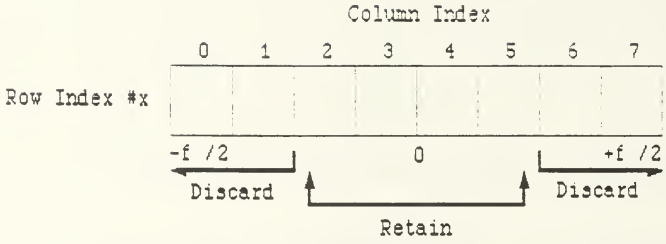

Figure 19 Retained Section of the Second FFT Results

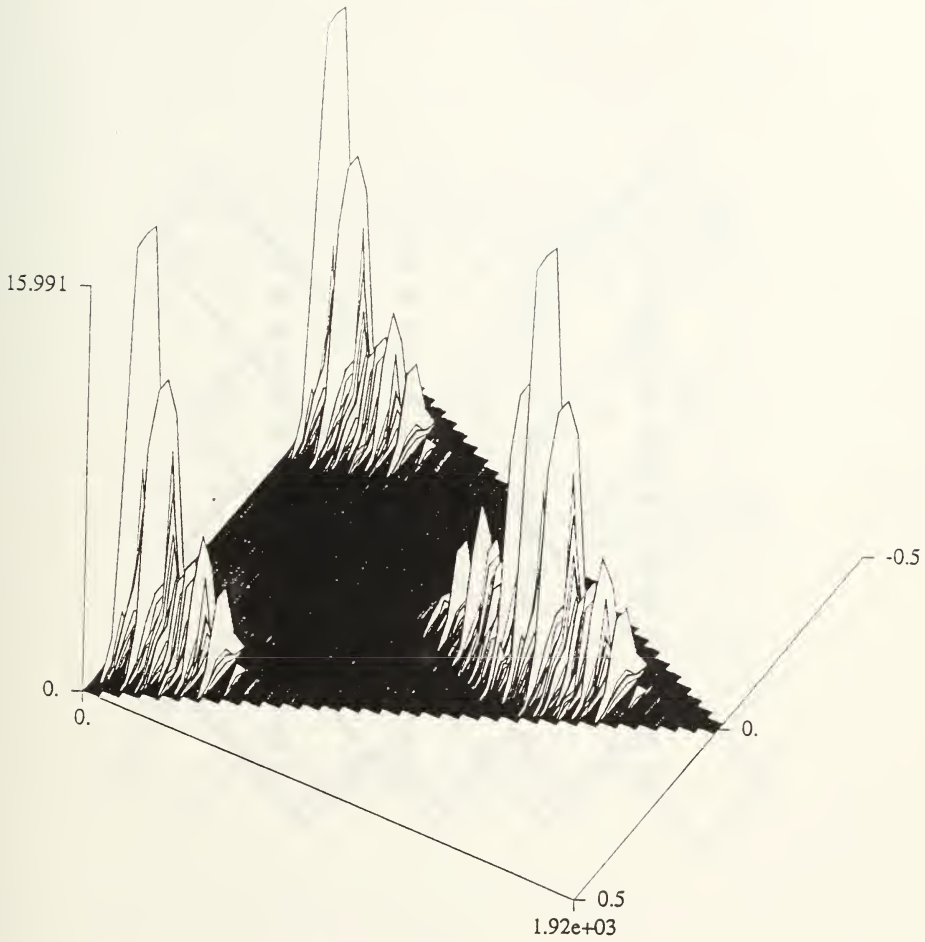

Figure 20 Final FAM Results of the BPSK Example, N=512, N'=32 and L=4.

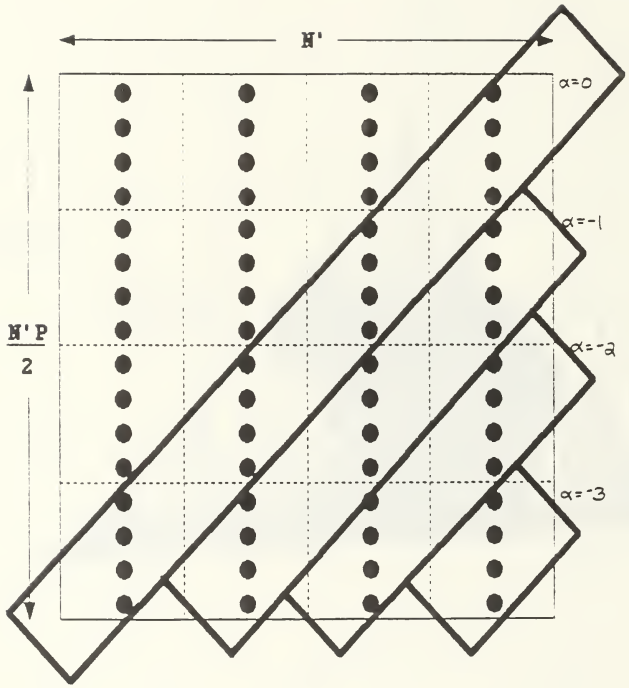

Figure 21 FAM Storage Before Data Reduction

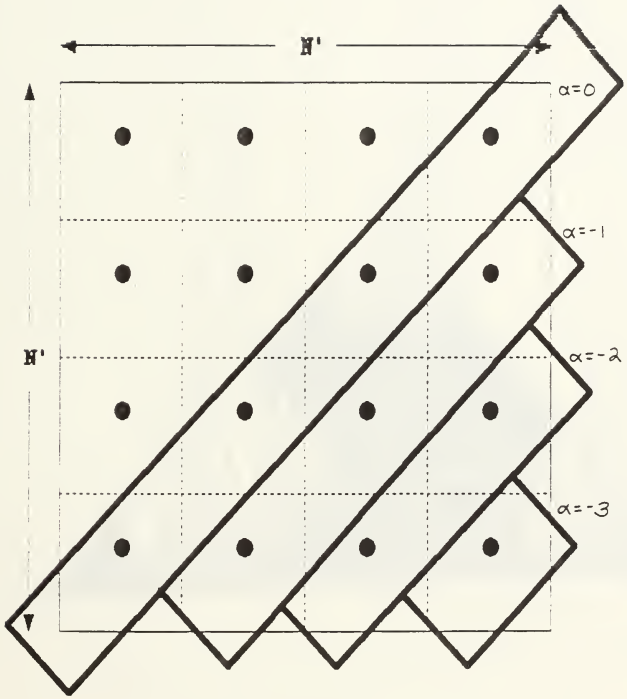

Figure 22 FAM Storage After Data Reduction

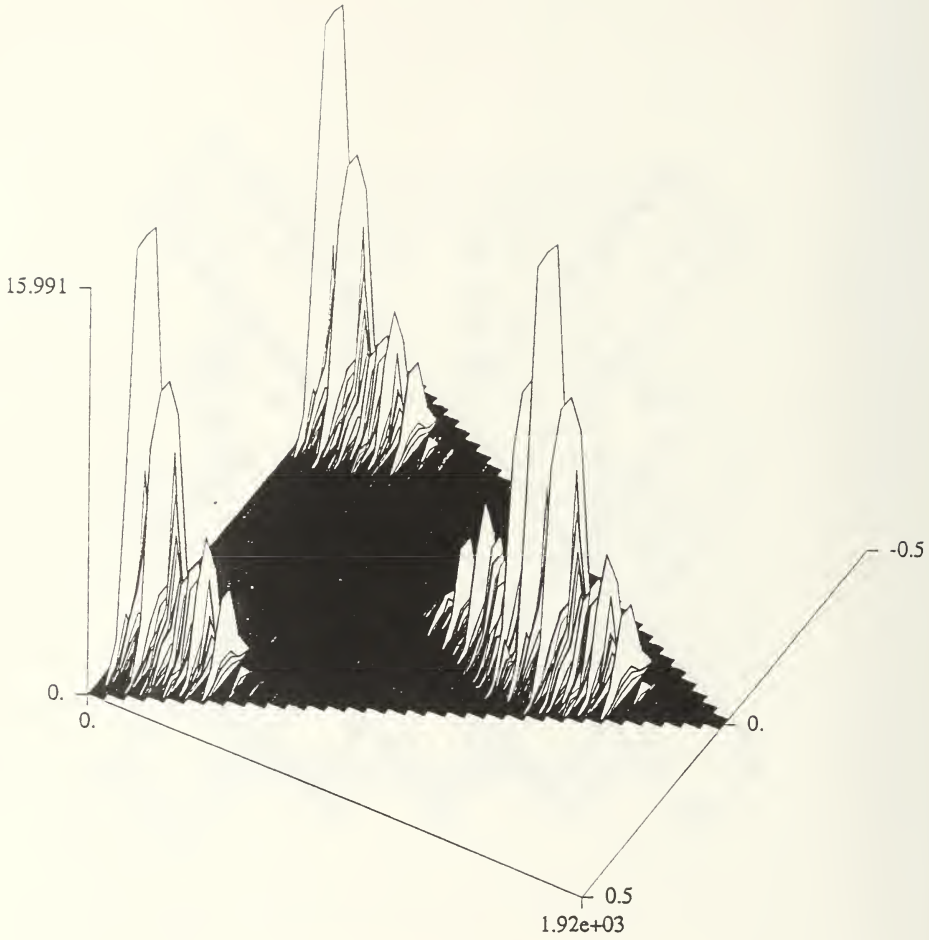

Figure 23 FAM Result Without Data Reduction, N=512, N'=32 and L=4.

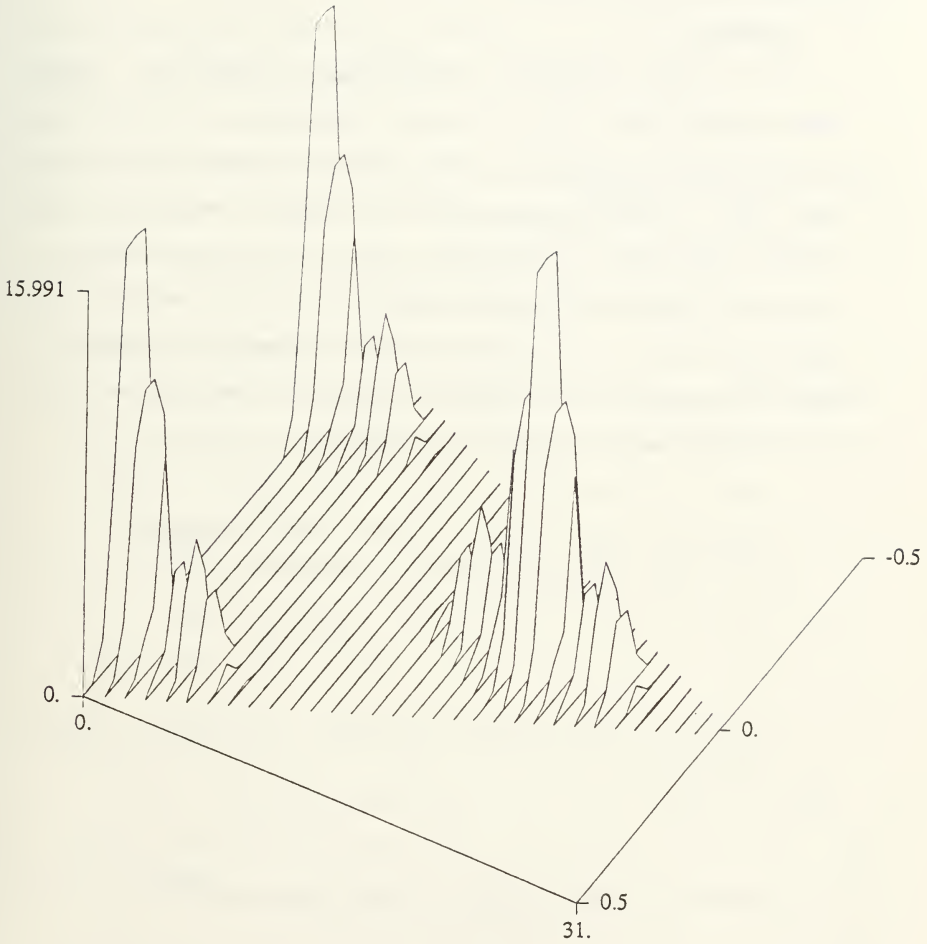

Figure 24 FAM Result With Data Reduction,

N=512, N'=32 and L=4.

III. STRIP SPECTRAL CORRELATION ALGORITHM

A. THEORY

The Strip Spectral Correlation Algorithm (SSCA) [Ref. 5:pp 47-48] is a Fourier transform of correlation products between spectral and temporal components smoothed over time. Periodicities in the spectral components then become detectable. The cyclic spectral plane as shown in Figure 25 ranges in frequency from $-f_s/2$ to $+f_s/2$, and in cycle frequency from $-f_s$ to $+f_s$ where f_s is the sampling frequency. For each unique (f_j, α_q) strip in Figure 25, the SSCA cross-spectral estimate is defined to be [Ref. 5:pg 47]:

$$S_{XY_T}^{f_k+q\Delta\alpha}\left(n, \frac{f_k}{2} - \frac{q\Delta\alpha}{2}\right)_{\Delta t} = \sum_{r=0}^{n-1} X_T(r, f_k)\, y^*(r)\, g(n-r)\, e^{\frac{-i2\pi rq}{n}} \qquad (7)$$

The SSCA auto-spectral estimate is defined to be [Ref. 5:pg 47]:

$$S_{XX_T}^{f_k+q\Delta\alpha}\left(n, \frac{f_k}{2} - \frac{q\Delta\alpha}{2}\right)_{\Delta t} = \sum_{r=0}^{n-1} X_T(r, f_k)\, x^*(r)\, g(n-r)\, e^{\frac{-i2\pi rq}{n}} \qquad (8)$$

where: k is a multiple of the frequency resolution,

$$-N'/2 \leq k \leq (N'/2)-1$$

q is a multiple of the cycle frequency resolution,

$$-N' \leq q \leq (N'-1)$$

g(n-r) represents the Hamming window.

The SSCA was implemented by forming an array from x(kT) ($0 \leq k \leq N-1$) with rows which are N' points long from the input sample data. The starting point of each succeeding row is offset from the previous rows starting position by L samples. A Hamming window is applied across each row which is then Fast Fourier transformed and downconverted to baseband. The result at this point is a two-dimensional array with columns representing constant frequencies. Each row is transposed and replicated L times for a total of PL columns. Each column is then multiplied by a sample y(kT) ($0 \leq k \leq (PL-1)$). Each resultant row of the array is Fast Fourier transformed and placed into a strip of the final cyclic spectral plane at the appropriate location. Figure 26 shows a block diagram for the SSCA cross-spectral estimate.

The following subsections in this chapter discuss in detail what has been described in the previous paragraph. The cycle frequency resolution of SSCA is $\Delta\alpha = f_s/N$ [Ref. 5:pg 48] where f_s is the original data sampling rate and N is the number of input samples used. The frequency resolution of

SSCA is equal to the sampling rate divided by the number of channels N', $\Delta f = f_s/N'$ [Ref. 5:pp 42,48]. The time-frequency resolution product is $\Delta t\Delta f = N/N'$ [Ref. 5:pg 48].

The command line format for calling the SSCA program is provided in Appendix C. The SSCA source code listing is provided in Appendix D.

B. IMPLEMENTATION

1. Input Channelization

The input sample data is formed into a two-dimensional array. The array row length is equal to the number of input channels N'. For a given number of input sample points N, a row size of N', and a chosen offset L, there are $P = (N-N')/L \approx N/L$ rows formed. The choice of N' must take into consideration that ideally the time-frequency resolution product must be much greater than one [Ref. 5:pg 40]. N' should also be a power of 2 to avoid truncation or zero-padding in the FFT routines. L should be chosen to be less than or equal to $N'/4$.

The completely filled array is P rows by N' columns. Figure 27 shows how a small array is filled from a discretely sampled signal x(kT) when $N'=16$, P=8 and L=4. The number inside each cell represents the value of k used to index on x(kT) to fill that location in the array.

Figure 28 shows the magnitude of the original data for the example BPSK signal organized into the P by N' array where

N=512, N'=32, L=4 and P=128. The input data is assumed to be complex with a real and imaginary component to each sample point. Figure 29 shows a single row of the array. The phase changes of the BPSK are evident.

2. Windowing

A Hamming window [Ref 7:pg 467] is applied to each row of the array. The equation for the Hamming window is:

$$w(r) = 0.54 - 0.46\cos\left(\frac{2\pi r}{N'-1}\right), \quad 0 \le r \le N'-1 \qquad (9)$$

A 32-point Hamming window is plotted in Figure 30. It is applied to both the real and imaginary parts of the complex example array. The magnitude of the resultant array is shown in Figure 31.

3. First FFT

Each row of the windowed data array is Fast Fourier transformed to reveal the first spectral components. The resultant array is still indexed P rows by N' columns but now the column index relates to a specific bin of spectral frequencies. Figure 32 illustrates this relationship. Figure 33 shows the results of FFTing the BPSK example.

4. Downconversion

Each row of spectral components is downconverted to baseband through multiplication with the complex exponential,

$$e^{\frac{-i2\pi kmL}{N'}}$$

where: m is the row index, $0 \leq m \leq P-1$

k is the column index, $0 \leq k \leq N'-1$

The magnitude of the exponential is unity over the array but the phase shows considerable variation. Figure 34 shows the phase of the exponential over the P by N' array. Figure 35 shows the phase for one representative row. The magnitude of the array remains unchanged from Figure 33.

5. Replication

Each row is copied into one column of an empty N' by PL array. It is then replicated in the L-1 adjacent columns. Figure 36 illustrates this process.

6. Multiplication

Each column of the array is pointwise multiplied with the conjugate of a sample value y(kT). There are PL≈N columns and PL samples from y(kT). Figure 37 illustrates the conjugate multiplication process.

7. Second FFT

Each row from the previous multiplication is Fast Fourier transformed to yield a PL-point result. Each resulting vector is stored into a strip of the cyclic spectral

plane. Figure 38 shows the final result of the SSCA auto-spectral process on the example BPSK signal.

8. Data Reduction

The SSCA Program typically generates large output data files. For convenience, an option may be chosen to reduce the amount of output. By comparison sorting for the largest α value in an f,α cell, the number of α slices is reduced from N to N/L. Overall SSCA execution time increases accordingly to accomplish the searches.

Figure 39 illustrates the output full cyclic spectral plane storage array before sorting. Figure 40 shows the output array after data reduction has been completed. Figures 41 and 42 plot the resulting spectral half-planes without and with data reduction respectively.

Since all cells have an equal number of α values, all sorts are of equal complexity. Further work on data reduction would require selection of a largest α value from widely varying numbers of α values depending on the choice of N, N' and L.

C. PERFORMANCE

An established method of evaluating the complexity of an algorithm is to determine the number of floating point operations that must be performed. For this estimate it is assumed that an auto-spectral estimate is being computed and the data is complex in every step. It is also assumed that

the Hamming window coefficients and the downconversion coefficients have been previously computed and stored for later use. Each N-point FFT requires $(N/2)*\log_2 N$ complex floating point multiplies [Ref. 8:pg 506] or $2*N*\log_2 N$ real floating point multiplies. The cost of any output data reduction is not considered here.

Applying the window.....$2*P*N'$

First FFT.....$2*P*N'*\log_2 N'$

Downconversion.....$4*P*N'$

Multiplication.....$4*N*N'$

Second FFT.....$2*N*N'*\log_2 N$

Total: $2*N'*(6*P + 4*N + (2*P + 2*N)*\log_2 N)$ (10)

since $P \approx N/L$ (11)

Total: $2*N'*((6*N/L + 4*N) + (2*N/L + 2*N)*\log_2 N)$ (12)

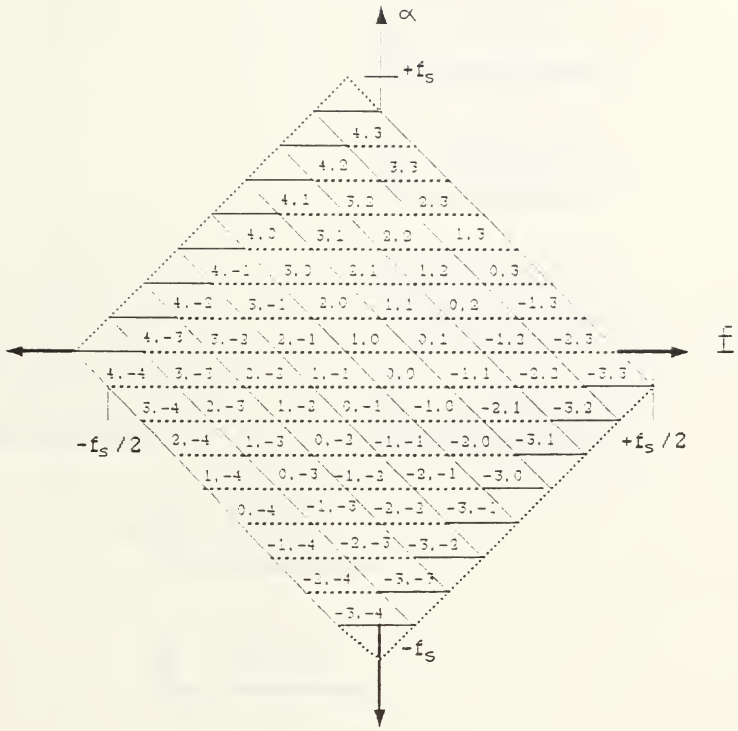

Figure 25 Generic SSCA Cyclic Spectral Plane

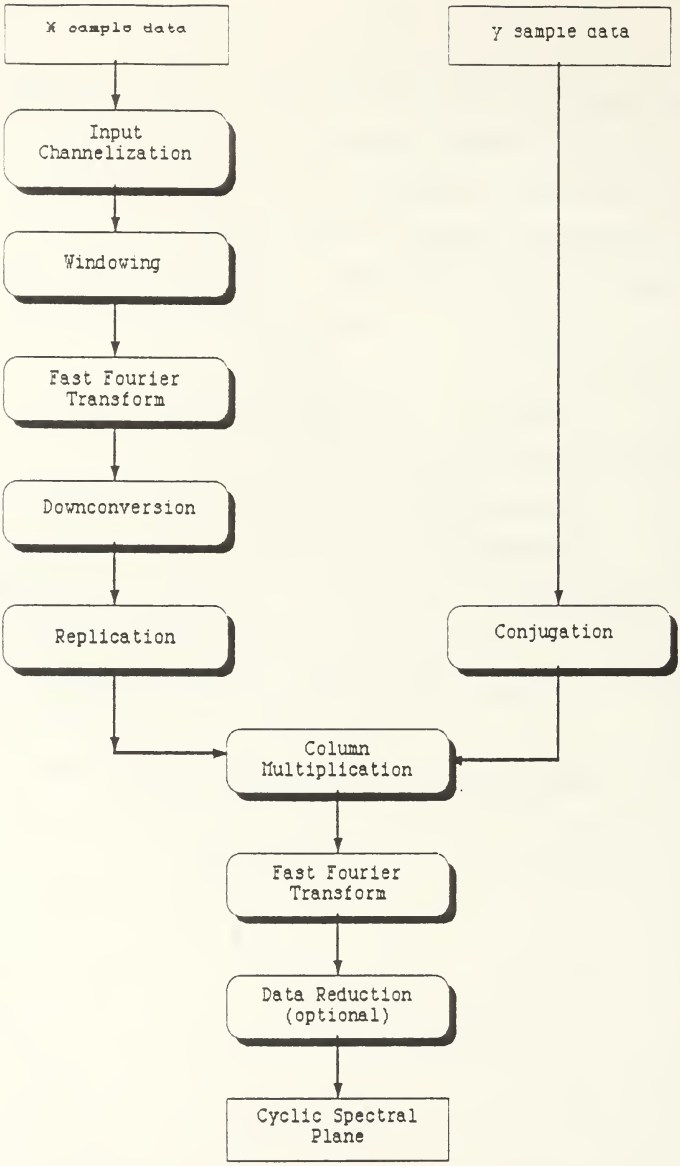

Figure 26 Block Diagram of SSCA Cross-Spectral Estimate

	Column Index															N'-1=
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
3	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
4	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
5	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
6	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
P-1=7	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

Figure 27 Layout of A Sample Input Array Showing Input Sample Storage for N'=16, N=48, L=4 and P=8.

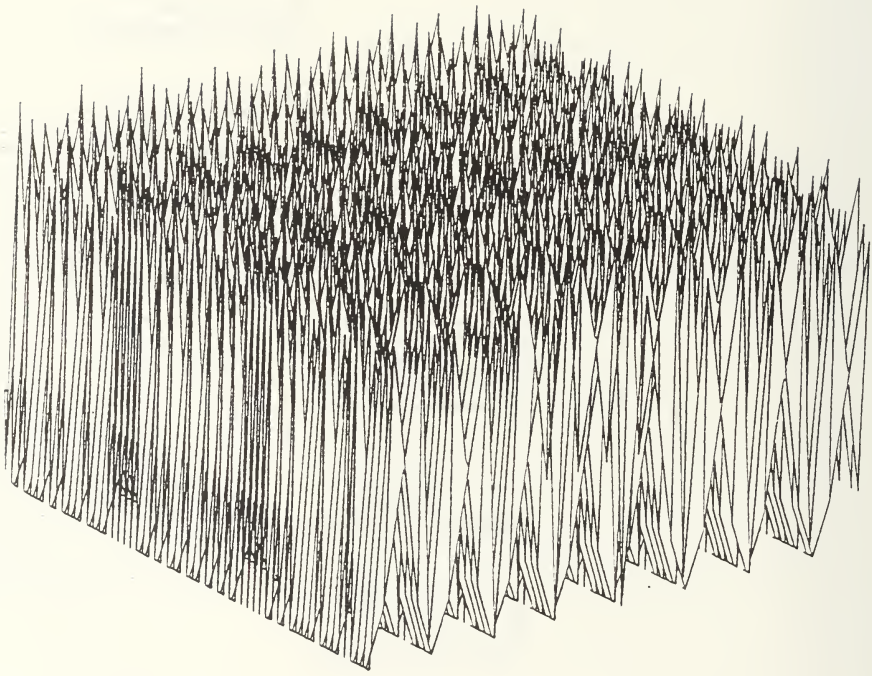

Figure 28 Example BPSK Signal Data Array, N'=32, P=128, L=4

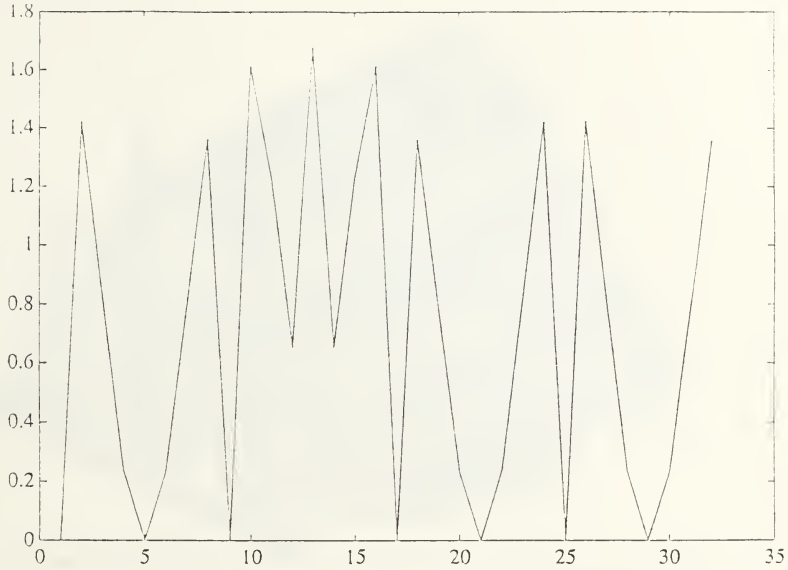

Figure 29 Single Row of BPSK Signal Input Array

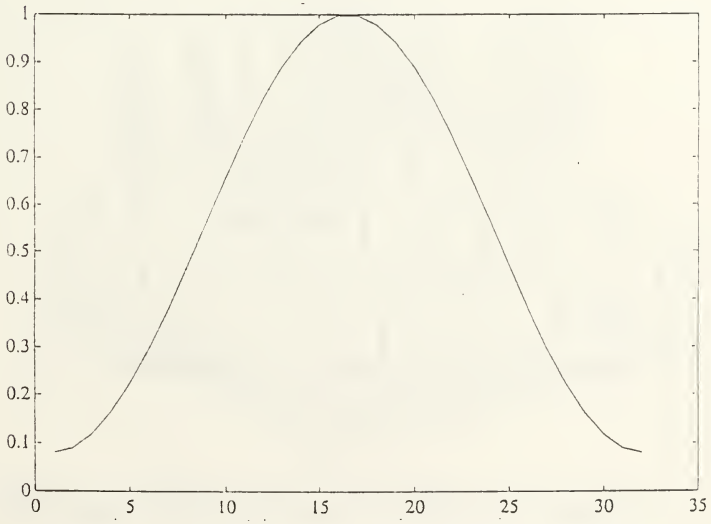

Figure 30 Thirty-two Point Hamming Window

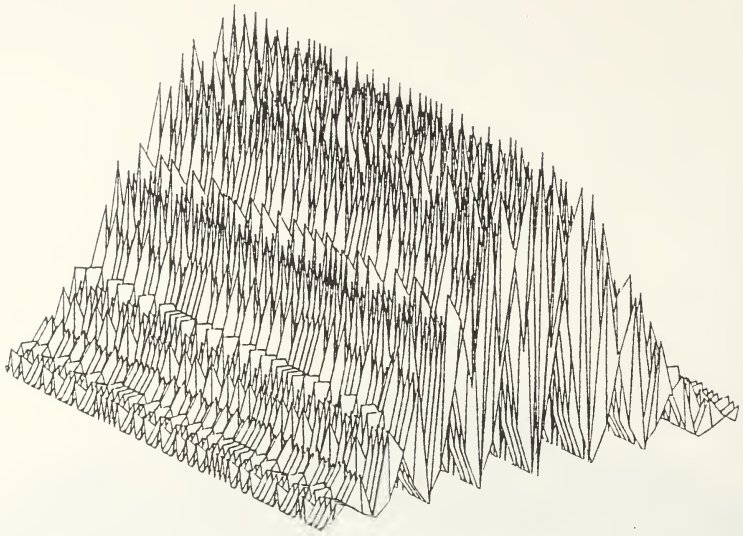

Figure 31 Example BPSK Signal Data After Windowing

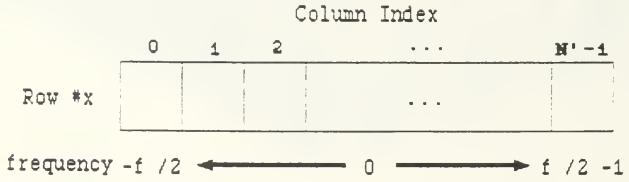

Figure 32 Generic Array Row After First FFT

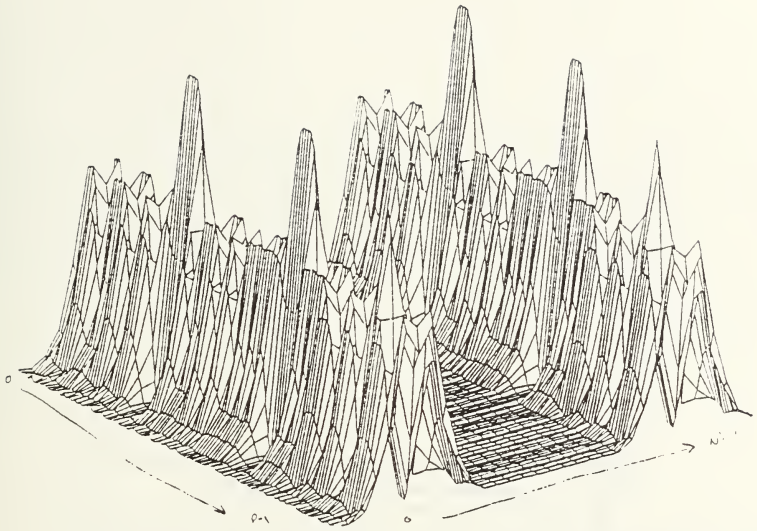

Figure 33 Example BPSK Signal Data After First FFT

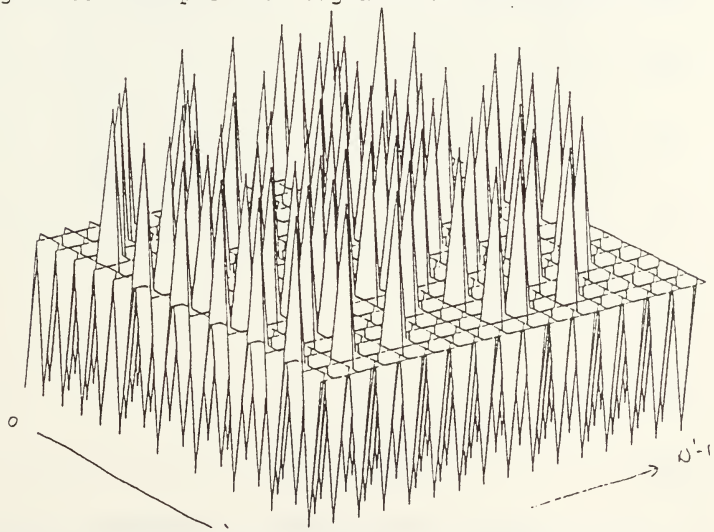

Figure 34 Phase of the Downconversion Exponential

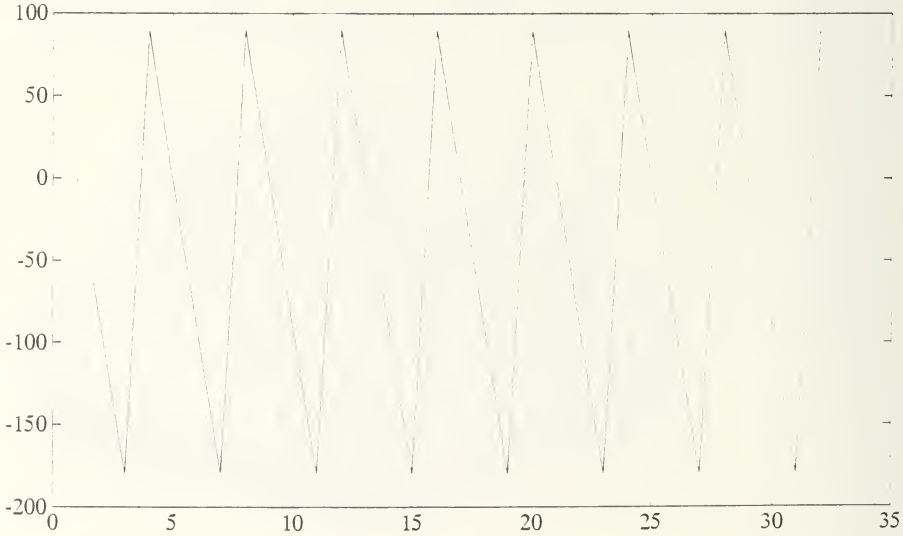

Figure 35 Phase For a Single Row of Array

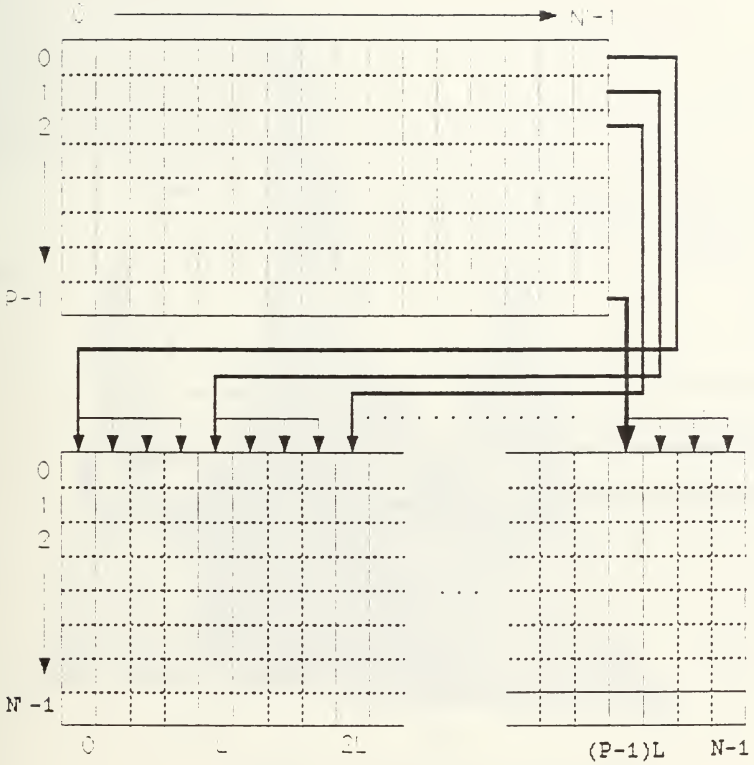

Figure 36 Replication Process

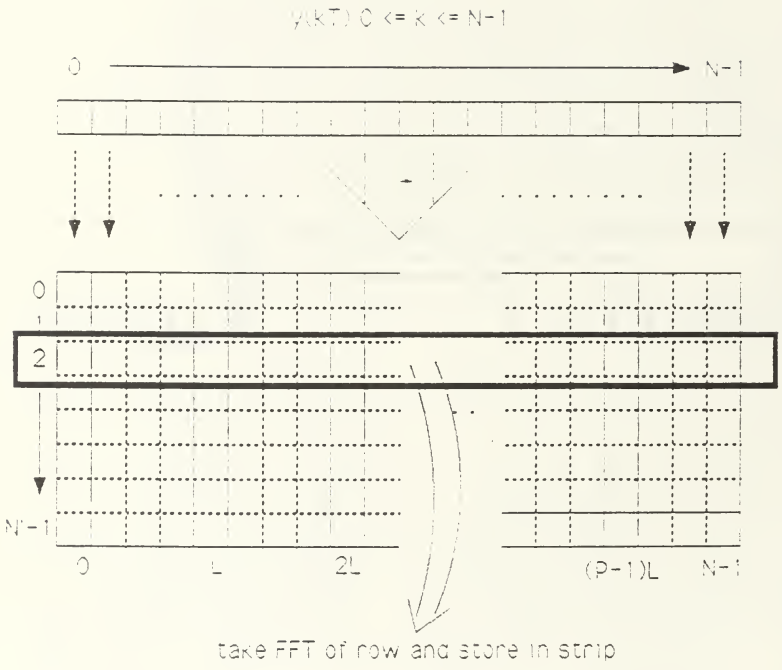

Figure 37 Multiplication of Columns by y*

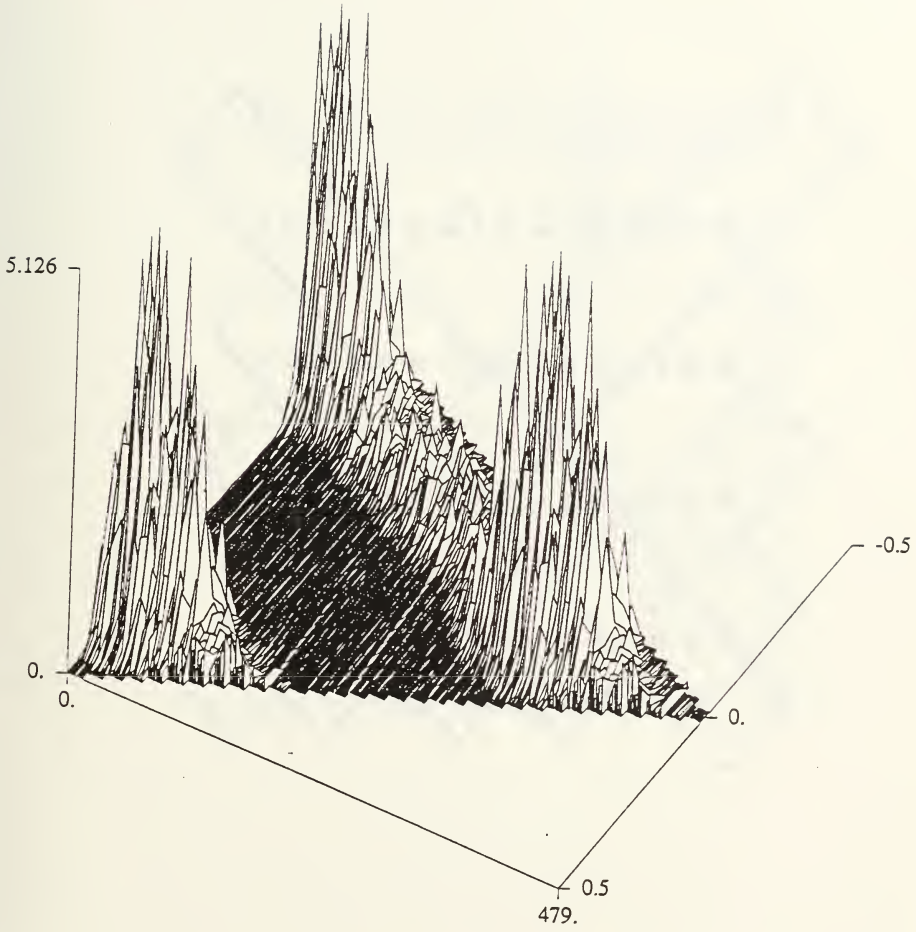

Figure 38 Final SSCA Results of the BPSK Example, N=512, N'=32 and L=4.

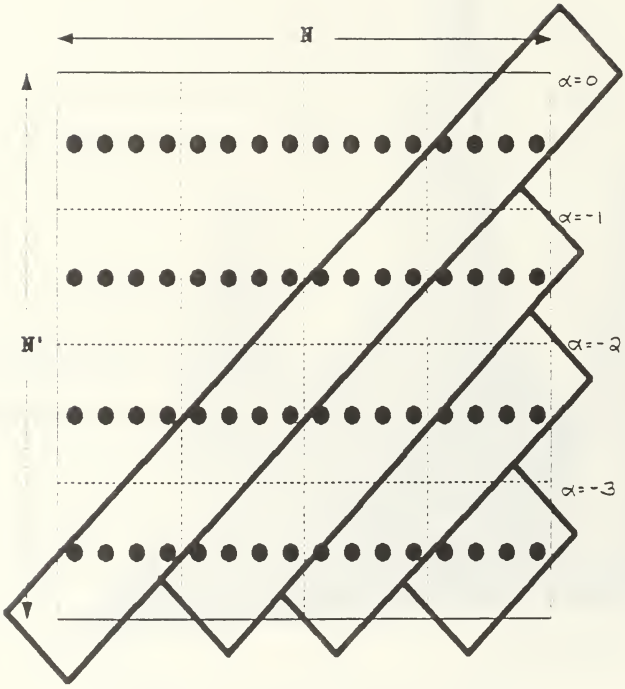

Figure 39 SSCA Storage Array Before Data Reduction

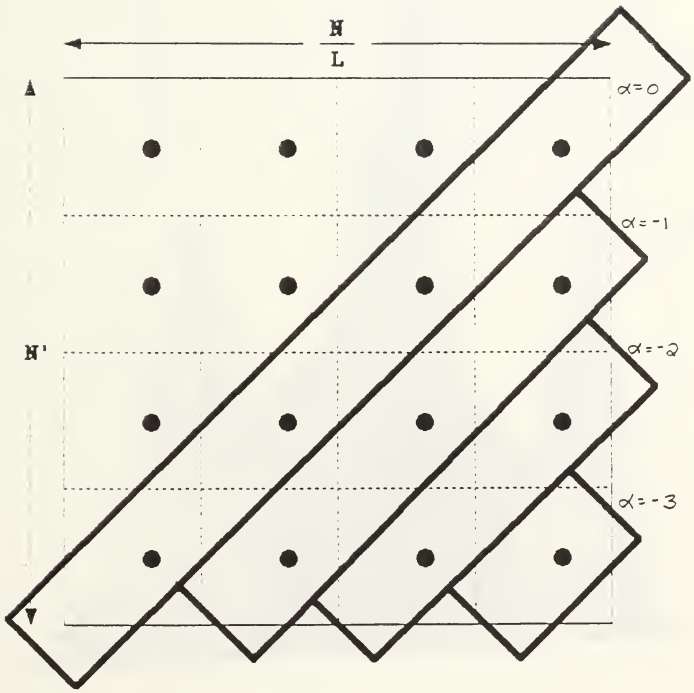

Figure 40 SSCA Storage Array After Data Reduction

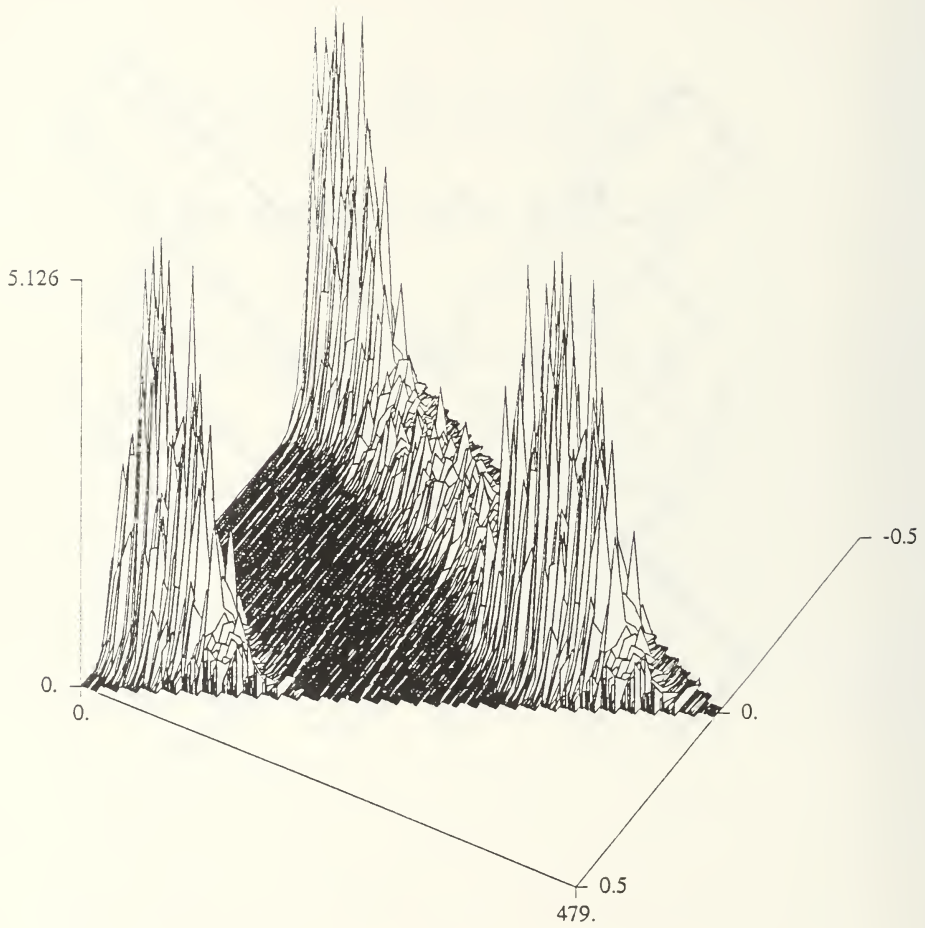

Figure 41 BPSK Example Results Without Data Reduction, N=512, N'=32 and L=4.

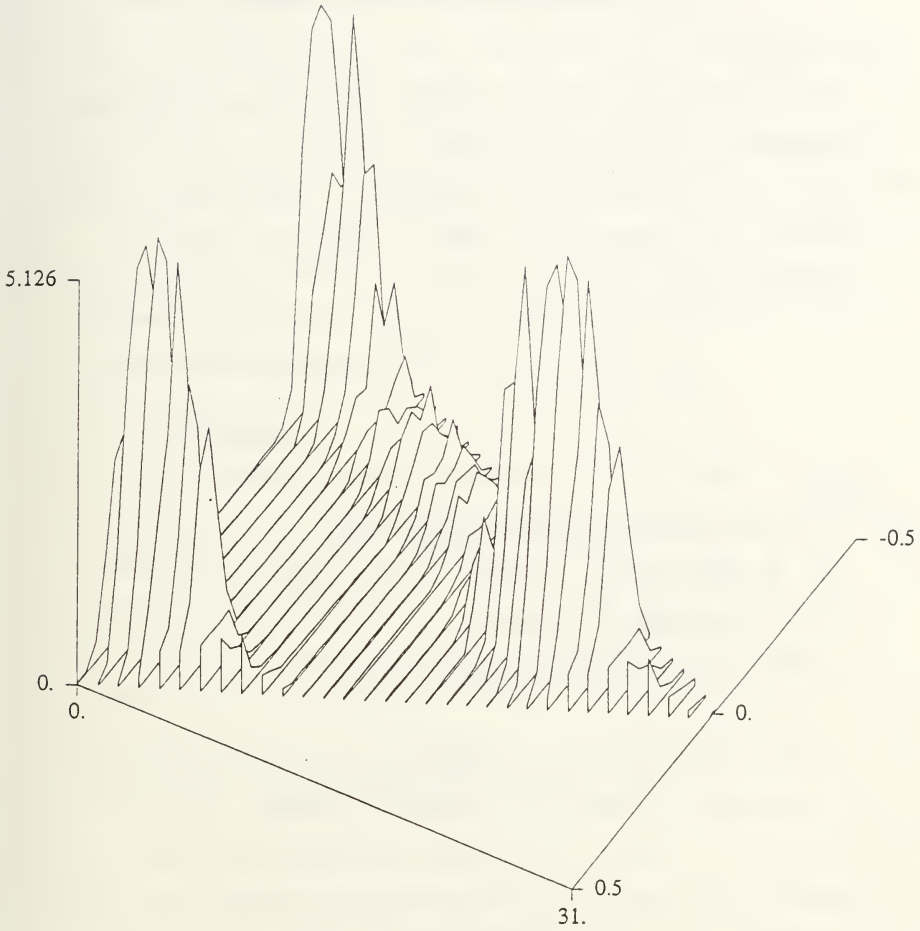

Figure 42 BPSK Example Results With Data Reduction, N=512, N'=32 and L=4.

IV. SUB-FFT ACCUMULATION METHOD

A. THEORY

In the course of this thesis work a need arose for a flexible and customized Spectral Correlation Function (SCF) [Ref. 7] to determine the presence of phase-shift keyed signals in experimental data. The SCF [Ref. 7] is represented:

$$S_{xy}^{\alpha}(f) = \sum_{r=0}^{N-1} x(r)\,y(r)\,e^{-i2\pi\frac{f_r}{f_s}r^2}\,OSF_x OSF_y e^{\left(-\frac{2\pi r}{N}\right)} \qquad (13)$$

where:

N is the number of data samples used

f_I is the intermediate modulation frequency

f_s is the sampling frequency

OSF is a one-sided bandpass filter

The program written to implement this algorithm is named the Sub-FFT Accumulation Method (SUBFAM) program to distinguish it from the more general purpose FAM program. The command line format for calling the SUBFAM program is provided in Appendix E. The SUBFAM source code is provided in Appendix F.

B. IMPLEMENTATION

Figure 43 illustrates the operations performed in the SUBFAM program. A set of N points is obtained from within a file of signal data values. Each set of N points is processed through a one-sided bandpass filter to remove either all positive or negative frequencies as desired. Downconversion of each file to baseband from the original sampling and transmission frequency bands follows. The results are then correlated through a complex conjugate multiplication. A final N-point Fast Fourier Transform yields the spectral correlation result.

C. PERFORMANCE

The SUBFAM program performed as required. Figure 44 illustrates the results of processing test data containing phase-shift-keyed signal samples. The peaks at the chip frequencies are apparent. Figure 45 shows the results of correlating test data with data containing only white noise. The lack of correlation between spectral features yields results which are four orders of magnitude less than in the successful signal detection of Figure 44.

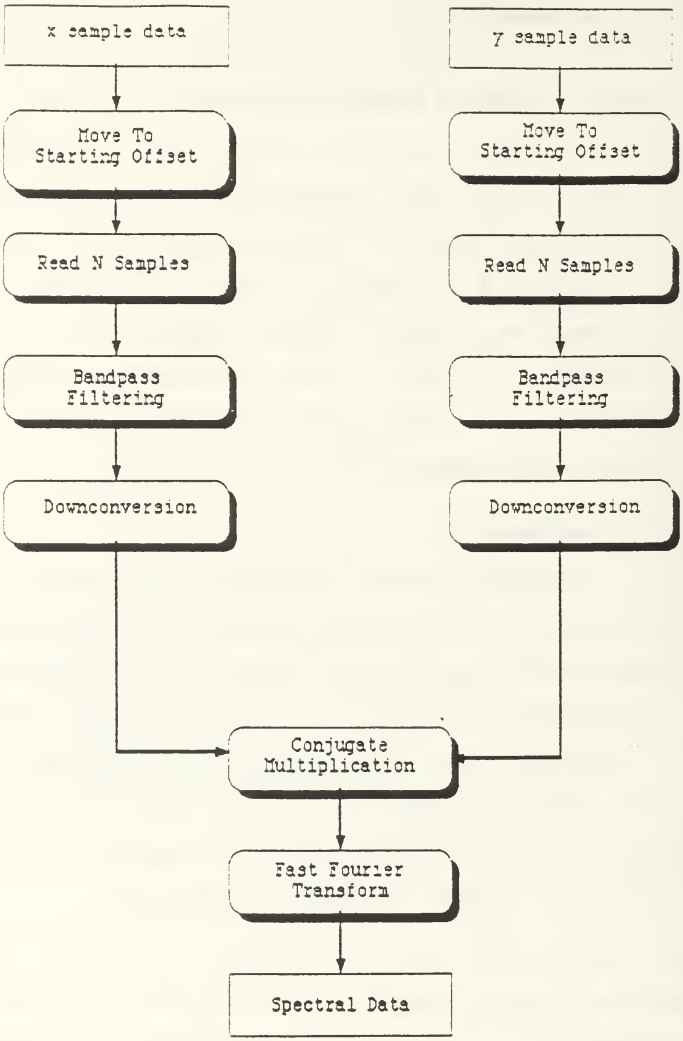

Figure 43 SUBFAM Program Block Diagram

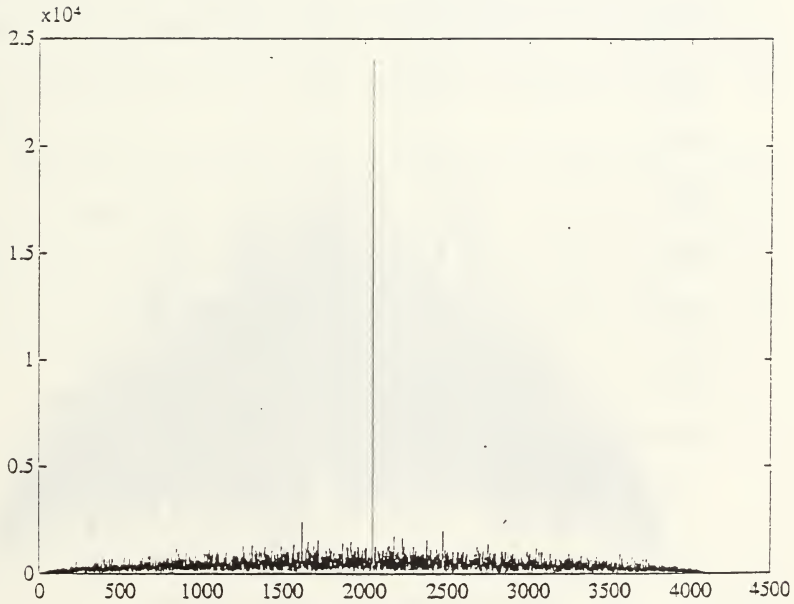

Figure 44 SUBFAM Program Result With Signal, N=4096

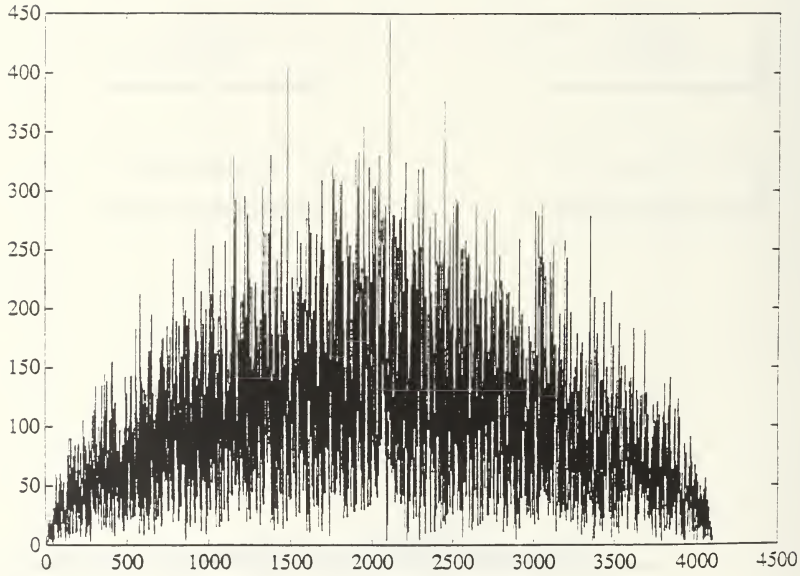

Figure 45 SUBFAM Program Result With Noise, N=4096

V. ALGORITHM PERFORMANCE

The complexity of the FAM and SSCA algorithms is estimated in Chapter II Section C and Chapter III Section C respectively. Both require on order of $N\log_2 N$ floating point multiplies where N is the original number of sample points used. Brown and Loomis [Ref. 9] explore the complexity of the FAM, SSCA and FSM algorithms in some detail. Their results are consistent with an order $N\log_2 N$ multiplies for FAM and SSCA.

The cyclic spectral plane is symmetric about the f=0 and the α=0 axes for real signals. For the most efficient performance it is only necessary to compute one quadrant of the cyclic spectral plane. To compare the performance of FAM and SSCA with FSM, complexity figures derived from Reference 9 for quadrant complexity will be used. The three algorithms require the following numbers of floating point multiplies:

FAM Complexity [Ref. 9:pg 18]

$$2NN'\log_2\left(\frac{4N}{N'}\right) + 8N\log_2(N') + 4NN' = 20N \qquad (14)$$

SSCA Complexity [Ref. 9:pg 20]

$$N\mathcal{N}'\log_2(N) + 2N\mathcal{N}' + 8N\log_2(\mathcal{N}') + 12N \qquad (15)$$

FSM Complexity [Ref. 9:pg 16]

$$N^2 + 2N\log_2(N) \qquad (16)$$

The complexity of four example cases with varying N and $\mathcal{N}'$ are estimated below using equation (14), (15) and (16).

N	N′	FAM	SSCA	FSM
1024	64	$5.3*10^5$	$3.6*10^5$	10^6
2048	128	$2.0*10^6$	$1.4*10^6$	$4.2*10^6$
32,768	512	$1.5*10^8$	$1.1*10^8$	10^9
1,048,576	8192	$8.0*10^{10}$	$7.0*10^{10}$	10^{12}

It is evident that FAM and SSCA require substantially fewer multiplies than FSM. Particularly with larger values of N, FAM and SSCA require $N\log_2 N$ while FSM requires N^2 floating point multiplies.

VI. CONCLUSIONS

A. SUMMARY

The purpose of this thesis was to implement the FAM and SSCA for use in cyclic spectral analysis research. Both algorithms were successfully implemented in a manner compatible with the SSPI analysis package to facilitate future work. By allowing the user a choice of two output file formats, results may easily be used in either MATLAB [Ref. 6] or SSPI [Ref. 4] for further signal processing applications.

It has been shown that the FAM and SSCA algorithms generate results consistent with FSM but require substantially fewer floating point multiplies than FSM. This is especially true as the number of sample data points used increases.

B. RECOMMENDATIONS

In the course of this thesis it has become apparent that the FAM and SSCA algorithms consist of inherently parallel operations. Roberts and Loomis explore these possibilities in Reference 10. The high-level sequential nature of these algorithms also lend them to pipelining architectures. By combining a parallel or multi-processor approach with pipelining a large improvement in performance should be obtainable. This would be espectially true in applications

requiring large sets of signal data samples or repeated, time-sequenced cyclic spectral plane estimations.

Two areas which could benefit from further work are automatic signal identification and signal parameterization algorithms. The results from cyclic spectral analysis algorithms such as FAM and SSCA may be input into Linear Associator or Mapping neurocomputer networks as described in Reference 11. Intensity transformations using digital image processing techniques from Reference 12 also appear to have promise. Both classes of techniques could have utility in automatic signal identification and parameterization. They also have the added benefit that they lend themselves to parallel and pipelined computing architectures.

APPENDIX A. FAM PROGRAM USE

Correct commands are:

for the cross-fam

```
fam inputfile1 inputfile2 outputfile n nprime l Oflag
fam inputfile1 inputfile2 outputfile n nprime l Oflag red
```

or for the auto-fam

```
fam inputfile1 outputfile n nprime l Oflag
fam inputfile1 outputfile n nprime l Oflag red
```

where: inputfile1 is the file containing the x signal samples
inputfile2 is the file containing the y signal samples
outputfile is where the spectrum values will be placed
n is the number of samples to use from the inputfiles
and it must be a power of 2
nprime is the group size of the input datasets and it
must be a power of 2
l is the starting offset of subsequent datasets and
it must be a power of 2
Oflag indicates the output filetype, ascii or binary
-asc for an ascii, MATLAB compatible file,
full cyclic spectral plane
-plo for a *plot_sxaf*, SSPI compatible file,
half cyclic spectral plane
red reduces the amount of data output

note: the first two entries in every input file are expected to be the datatype and n. datatype = 1 for real values, 2 for complex values. n is the number of samples contained in the file, one sample per line.

input file format:

```
datatype    n
sample #1
sample #2
   .
   .
   .
sample #n
```

output file ASCII MATLAB format:

```
nprime      n
output(1)(1)
output(1)(2)
   .
   .
   .
output(nprime)(n)
```

output file SSPI *plot_sxaf* format:

```
datatype (1 for real, 2 for complex)
number_of_alphasalpha_minalpha_max
number_of_freqsfreq_minfreq_max

value of alpha #1
number of freqs in #1freq_minfreq_max
spectrum at freq_min
   .
   .
   .
spectrum at freq_max

      .
      .
      .

value of alpha #alpha_max
number of freqs in #alpha_maxfreq_minfreq_max
spectrum at freq_min
   .
   .
   .
spectrum at freq_max
```

APPENDIX B. FAM PROGRAM LISTING

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "/home3/carter/thesis/SSPI/pam/fft.c"
#include "/home3/carter/thesis/SSPI/pam/radix.c"
main(argc,argv)
int argc;
char *argv[];
{
COMPLEX *x,**s3,**y3,**dwnconv,**s4;
COMPLEX *tempfft,*y1,*s1,**s2,**y2,*window;
/*  x passes data to/from fft routine
    s3 holds results of FFTing s2
    y3 holds results of FFTing y2
    dwnconv holds downconversion coefficients
    s4 holds correlation multiply results
    tempfft passes data to/from fft routine
    y1 holds y samples from inputfile 2
    s1 holds x samples from inputfile 1
    s2 holds channelized x samples
    y2 holds channelized y samples
    window holds Hamming window values
                                                */
int i,j,k,type,n,file_n,nprime,p,l,direction,norm;
int a,b,c,cross,num_f,max_num_f,data_type,max_num_alf;
int tempi,tempj,halfp,reduce,curr_alf;
int *outint;
float numreal,numimag,convfac,mainfac,num_alf,f_min,f_max;
float alpha_min,alpha_max,f_min_all,f_max_all;
float tempreal,tempimag,bigmag,tempmag;
float twopi=6.28318530718;
float *outreal;
double z,y;
char *infile1,*infile2,*outfile;
FILE *ifp1,*ifp2,*ofp;
/* check for the correct number of input arguments
   the correct commands are:
        fam inputfile1 inputfile2 outputfile n nprime l Oflag reduce
            or
        fam inputfile1 inputfile2 outputfile n nprime l Oflag
            or
        fam inputfile1 outputfile n nprime l Oflag reduce
            or
        fam inputfile1 outputfile n nprime l Oflag

   where:  inputfile1 is where the x signal samples are expected to be
           inputfile2 is where the y signal samples are expected to be
           outputfile is where the spectrum values will be put
           n is the number of samples to use from the inputfiles
           nprime is the group size of input datasets
           l is the starting offset of subsequent datasets
           Oflag indicates the output filetype, ascii or binary
              -asc  for an ascii, matlab compatible file, full plane
              -plo  for a plot_sxaf compatible file, half plane
           reduce is an option to reduce the amount of output
```

```
   note:  datatype and n, are expected to be at the top of the input file
            datatype = 1 for real, 2 for complex
            n is the number of samples following

   written by LCDR Nancy J. Carter, USN    NPGS Monterey CA  01 October 1992
   to implement the Frequency Accumulation Method of cyclic spectral analysis
   as described in the Brown/Gardner/Loomis IEEE paper

   20 October 1992....changed to ascii MATLAB output format
   22 October 1992....added plot_sxaf output format compatible with SSPI
   23 November 1992....added option to reduce amount of output
                                                                        */
/*    CHECK INPUT VALUES    */
/*                          */
if ((argc != 7)&&(argc != 8)&&(argc != 9)){
  printf("fam....fatal error\n");
  printf(".......incorrect number of calling arguments\n");
  printf(".......correct formats are:\n");
  printf(".........  fam inputfile1 inputfile2 outputfile n nprime l Oflag reduce\n");
  printf(".........  fam inputfile1 inputfile2 outputfile n nprime l Oflag\n");
  printf(".........  fam inputfile outputfile n nprime l Oflag reduce\n");
  printf(".........  fam inputfile outputfile n nprime l Oflag\n");
  printf(".........    where\n");
  printf(".........      inputfile1 contains the x samples\n");
  printf(".........      inputfile2 contains the y samples\n");
  printf(".........      outputfile will contain the results\n");
  printf(".........      n is the number of input samples to use\n");
  printf(".........      nprime is the group size of input datasets\n");
  printf(".........      l is the offset of subsequent datasets\n");
  printf(".........      Oflag is the output file format\n");
  printf(".........          Oflag = -asc, an ascii file is produced\n");
  printf(".........          Oflag = -plo, a plot_sxaf file is produced\n");
  printf(".........      reduce is an option for reduced amount of output\n");
  exit();
  }
if (argc==7) {  /* this is an auto-fam, no output reduction  */
  cross=0;
  infile1=argv[1];
  outfile=argv[2];
  n=atoi(argv[3]);
  nprime=atoi(argv[4]);
  l=atoi(argv[5]);
  reduce=0;
  }
if ((argc==8)&&(argv[7][0]!='r')) {  /* this is a cross-fam, no output reduction  */
  cross=1;
  infile1=argv[1];
  infile2=argv[2];
  outfile=argv[3];
  n=atoi(argv[4]);
  nprime=atoi(argv[5]);
  l=atoi(argv[6]);
  reduce=0;
  }
if ((argc==8)&&(argv[7][0]=='r')) {  /* this is an auto-fam, with output reduction  */
  cross=0;
```

```
  infile1=argv[1];
  outfile=argv[2];
  n=atoi(argv[3]);
  nprime=atoi(argv[4]);
  l=atoi(argv[5]);
  reduce=1;
  }
if (argc==9) {  /* this is a cross-fam, with output reduction  */
  cross=1;
  infile1=argv[1];
  infile2=argv[2];
  outfile=argv[3];
  n=atoi(argv[4]);
  nprime=atoi(argv[5]);
  l=atoi(argv[6]);
  reduce=1;
  }
/* verify that n, nprime and l are powers of 2      */
i=n%2;
if (i!=0){
  printf("fam....fatal error\n");
  printf(".......calling argument n is not a power of 2\n");
  exit();
  }
i=nprime%2;
if (i!=0){
  printf("fam....fatal error\n");
  printf(".......calling argument nprime is not a power of 2\n");
  exit();
  }
j=l%2;
if (j!=0){
  printf("fam....fatal error\n");
  printf(".......calling argument l is not a power of 2\n");
  exit();
  }
/*                                                              */
/* open input file 1, prepare to get the x signal sample data   */
/*                                                              */
ifp1 = fopen(infile1,"r");
fscanf(ifp1,"%i %i",&type,&file_n);
/* verify that file_n is greater than or equal to n */
if (file_n<n){
  printf("fam....fatal error\n");
  printf(".......inputfile1 does not contain enough samples\n");
  fclose(ifp1);
  exit();
  }
/* find p - the number of datasets(nprime long)  */
p=((n-nprime)/l);
/*                                                              */
/*    */
/*    READ IN DATA SAMPLES   */
/*    */
/* allocate space to read in the sample values */
/*    */
```

```
s1=(COMPLEX*)calloc(n,sizeof(COMPLEX));
if (s1==NULL){
  printf("fam....insufficient space to allocate s1\n");
  exit();
  }
if (type==1) {
/*                                                              */
/*  read in the real sample values */
/*                                                              */
 for (i=0; i < n; i++){
        fscanf(ifp1,"%e\n",&numreal);
        s1[i].r=numreal;
        s1[i].i=0.0;
 }
}
else {
/*                                                              */
/*  read in the complex sample values */
/*                                                              */
 for (i=0; i < n; i++){
        fscanf(ifp1,"%e  %e\n",&numreal,&numimag);
        s1[i].r=numreal;
        s1[i].i=numimag;
 }
}
/*    */
/*  close the input file #1   */
/*    */
fclose(ifp1);
/*    */
/* if this is a cross-fam go get y samples  */
/*    */
if (cross==1) {
/*                                                              */
/* open inputfile2, prepare to get the y signal sample data   */
/*                                                              */
  ifp2 = fopen(infile2,"r");
  fscanf(ifp2,"%i %i",&type,&file_n);
  /* verify that file_n is greater than or equal to n */
  if (file_n<n){
    printf("fam....fatal error\n");
    printf(".......inputfile1 does not contain enough samples\n");
    fclose(ifp1);
    exit();
    }
/*    */
/*    READ IN Y DATA SAMPLES   */
/*    */
/* allocate space to hold the sample values */
/*    */
  y1=(COMPLEX*)calloc(n,sizeof(COMPLEX));
  if (y1==NULL){
    printf("fam....insufficient space to allocate y1\n");
    exit();
    }
/*                                                              */
```

```
/*  read in the complex sample values */
/*                                                                  */
  for (i=0; i < n; i++){
    fscanf(ifp2,"%e  %e\n",&numreal,&numimag);
    y1[i].r=numreal;
    y1[i].i=numimag;
  }
/*    */
/*  close the inputfile2   */
/*    */
  fclose(ifp2);
}  /* end if cross=1   */
/*    */
/*    FORM DATASETS    */
/*    */
/* allocate space to hold the p-by-nprime datasets  */
/*    */
s2=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (s2==NULL){
  printf("fam....insufficient space to allocate s2\n");
  exit();
  }
for (i=0; i<p; i++){
  s2[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (s2[i]==NULL){
    printf("fam....insufficient space to allocate s2\n");
    exit();
    }
  }
/*                                                                  */
/* form p datasets of nprime samples from the original sample stream  */
/*                                                                  */
for (i=0; i<p; i++){
  for (j=0; j<nprime; j++){
    k=i*l+j;
    s2[i][j]=s1[k];
  }
}
/*    */
/* if this is a cross-fam form y datasets */
/*    */
if (cross==1) {
/*                                                                */
/*    */
/* allocate space to hold the p-by-nprime datasets  */
/*    */
  y2=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
  if (y2==NULL){
    printf("fam....insufficient space to allocate y2\n");
    exit();
    }
  for (i=0; i<p; i++){
    y2[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
    if (y2[i]==NULL){
      printf("fam....insufficient space to allocate y2\n");
      exit();
```

```
      }
    }
/*                                                              */
/* form p datasets of nprime samples from the original sample stream  */
/*                                                              */
  for (i=0; i<p; i++){
    for (j=0; j<nprime; j++){
      k=i*l+j;
      y2[i][j]=y1[k];
    }
  }
}  /* end if cross=1   */
/*    */
/*    APPLY WINDOW TO DATASETS    */
/*    */
/* allocate space to hold the window multiplicand values */
/*    */
window=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
if (window==NULL){
  printf("fam....insufficient space to allocate s1\n");
  exit();
  }
/*                                              */
/* calculate the window values for the "nprime" sample wide window  */
/*                                              */
for (i=0; i<nprime; i++){
    y=(twopi*i)/(nprime-1);
    z=cos(y);
    window[i].r= 0.54 - (0.46*z);
    window[i].i= window[i].r;
}
/*                                              */
/* apply the window to all rows of s2           */
/*                                              */
for (i=0; i<p; i++){
    for (j=0; j<nprime; j++){
        s2[i][j].r=s2[i][j].r*window[j].r;
        s2[i][j].i=s2[i][j].i*window[j].i;
    }
}
/*    */
/* if this is a cross-fam apply window to y2 */
/*    */
if (cross==1) {
/*                                                              */
  for (i=0; i<p; i++){
    for (j=0; j<nprime; j++){
      y2[i][j].r=y2[i][j].r*window[j].r;
      y2[i][j].i=y2[i][j].i*window[j].i;
    }
  }
}  /* end if cross=1   */
/*    */
/*  FFT EACH DATASET ROW    */
/*    */
/*  allocate space to hold rows of data for passing to the FFT routine */
```

```
/*    */
x=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
if (x==NULL){
  printf("fam....insufficient space to allocate x\n");
  exit();
  }
/*    */
/*  allocate space to hold rows of results from the FFT routine */
/*    */
s3=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (s3==NULL){
  printf("fam....insufficient space to allocate s3\n");
  exit();
  }
for (i=0; i<p; i++){
  s3[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (s3[i]==NULL){
    printf("fam....insufficient space to allocate s3\n");
    exit();
    }
  }
/*                                                  */
/* take an FFT of each row of s2                    */
/*                                                  */
/*    set values in arguments sent to fft  */
direction=1;
norm=1;
for (i=0; i<p; i++){
  /*    copy row of s2 into complex array      */
  for (j=0; j<nprime; j++){
    x[j]=s2[i][j];
    }
  /*    go get FFT performed                    */
  fft(x,nprime,direction,norm);
  /* copy x values into a row of s3             */
  for (j=0; j<nprime; j++){
    s3[i][j]=x[j];
    }
}
/*    */
/* if this is a cross-fam take an FFT of each row of y2 */
/*    */
if (cross==1) {
/*                                                                  */
/*    */
/*  allocate space to hold rows of results from the FFT routine */
/*    */
y3=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (y3==NULL){
  printf("fam....insufficient space to allocate y3\n");
  exit();
  }
for (i=0; i<p; i++){
  y3[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (y3[i]==NULL){
    printf("fam....insufficient space to allocate y3\n");
```

```
    exit();
    }
}
 for (i=0; i<p; i++){
  /*    copy row of y2 into complex array      */
  for (j=0; j<nprime; j++){
    x[j]=y2[i][j];
    }
  /*    go get FFT performed                   */
  fft(x,nprime,direction,norm);
  /* copy x values into a row of y3           */
  for (j=0; j<nprime; j++){
    y3[i][j]=x[j];
    }
  }
}  /* end if cross=1   */
/*    */
/*    */
/*    DOWNCONVERT EACH ROW    */
/*    */
/*  allocate space to hold the downconversion multiplicands  */
/*    */
dwnconv=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (dwnconv==NULL){
  printf("fam....insufficient space to allocate dwnconv\n");
  exit();
  }
for (i=0; i<p; i++){
  dwnconv[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (dwnconv[i]==NULL){
    printf("fam....insufficient space to allocate dwnconv\n");
    exit();
    }
  }
/*                                          */
/*  downconvert each of the transform sequences    */
/*                                          */
/*      calculate the down conversion multipliers    */
/*                                          */
mainfac=twopi*1/nprime;
for (i=0; i<p; i++){
  for (j=0; j<nprime; j++){
    convfac=i*j*mainfac;
    dwnconv[i][j].r=cos(convfac);
    dwnconv[i][j].i= (-1.0)*sin(convfac);
  }
}
/* multiply downconversion factor against each frequency value  */
  for (i=0; i<p; i++){
  for (j=0; j<nprime; j++){
    numreal=s3[i][j].r*dwnconv[i][j].r - s3[i][j].i*dwnconv[i][j].i;
    s3[i][j].i=s3[i][j].r*dwnconv[i][j].i + s3[i][j].i*dwnconv[i][j].r;
    s3[i][j].r=numreal;
  }
}
if (cross==1) {
```

```
/*    */
/* if this is a cross-fam downconvert y3 */
/*    */
/* multiply downconversion factor against each frequency value  */
  for (i=0; i<p; i++){
    for (j=0; j<nprime; j++){
      numreal=y3[i][j].r*dwnconv[i][j].r - y3[i][j].i*dwnconv[i][j].i;
      y3[i][j].i=y3[i][j].r*dwnconv[i][j].i + y3[i][j].i*dwnconv[i][j].r;
      y3[i][j].r=numreal;
    }
  }
}  /* end if cross=1  */
/*    */
/*    MULTIPLY COLUMNS    */
/*    */
/*  allocate space to hold the correlation multiply (column multiply) results */
/*    */
a=nprime*p/2;
s4=(COMPLEX**)calloc(a,sizeof(COMPLEX*));
if (s4==NULL){
  printf("fam....insufficient space to allocate s4\n");
  exit();
  }
for (i=0; i<a; i++){
  s4[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (s4[i]==NULL){
    printf("fam....insufficient space to allocate s4\n");
    exit();
    }
  }
/*    */
/*  allocate space to hold columns of s4 for passing to the FFT routine */
/*    */
tempfft=(COMPLEX*)calloc(p,sizeof(COMPLEX));
if (tempfft==NULL){
  printf("fam....insufficient space to allocate tempfft  %i by 1\n",p);
  exit();
  }
/*    set values in arguments sent to fft  */
direction=1;
norm=1;
if (cross==0)  {  /* auto-fam  */
/*    */
/*  multiply columns of s3  */
/*    */
for (a=nprime-1; a> -1; a--) {
  for (b=0; b<nprime; b++)  {
/*  multiply the columns of s3 with other columns of s3   */
    for (j=0; j<p; j++){
      tempfft[j].r=(s3[j][a].r*s3[j][b].r)+(s3[j][a].i*s3[j][b].i);
      tempfft[j].i=(s3[j][a].i*s3[j][b].r)-(s3[j][a].r*s3[j][b].i);
    }
/*                                          */
/*   FFT EACH COLUMN     */
/*    */
  /*    go get FFT performed                  */
```

```
  fft(tempfft,p,direction,norm);
  /* copy tempfft result values back into a row of s4          */
  for (j=(p/4); j<((3*p/4)-1); j++){
    c=((nprime-a-1)*p/2)+j-(p/4);
    s4[c][b]=tempfft[j];
    }
  } /* end for b=0... */
} /* end for a=0...  */
}  /* end auto-fam  */
else {  /* cross-fam  */
/*    */
/*  multiply columns of s3 with y3  */
/*    */
for (a=nprime-1; a> -1; a--) {
  for (b=0; b<nprime; b++)  {
/*  multiply the columns of s3 with other columns of y3   */
    for (j=0; j<p; j++){
      tempfft[j].r=(s3[j][a].r*y3[j][b].r)+(s3[j][a].i*y3[j][b].i);
      tempfft[j].i=(s3[j][a].i*y3[j][b].r)-(s3[j][a].r*y3[j][b].i);
    }
/*                                                 */
/*   FFT EACH COLUMN     */
/*    */
  /*    go get FFT performed                   */
  fft(tempfft,p,direction,norm);
  /* copy tempfft result values back into a row of s4          */
  for (j=(p/4); j<((3*p/4)-1); j++) {
    c=((nprime-a-1)*p/2)+j-(p/4);
    s4[c][b]=tempfft[j];
    }

  } /* end for b=0... */
} /* end for a=0...  */
}  /* end cross-fam  */
/*    */
/*  OUTPUT  */
/*    */
halfp=p/2;
if (reduce==0) { /* do not reduce the amount of output */

  if (((argc==8)&&(argv[7][1]=='a'))||((argc==7)&&(argv[6][1]=='a'))) {
  /* make an ascii output file of the whole spectral plane */
  /* open the output file and place header information in it */
     ofp = fopen(outfile,"w");
     max_num_alf=nprime*p/2;
     max_num_f=nprime;
     fprintf(ofp,"%i %i\n",max_num_alf,max_num_f);
     for (i=0; i<max_num_alf; i++){
       for (j=0; j<nprime; j++){
         fprintf(ofp,"%e  %e\n",s4[i][j].r,s4[i][j].i);
       }  /* end for j=...  */
     }  /* end for i=...   */
     fclose(ofp);
  }  /* end if ascii...  */

  else { /* make a plot_sxaf no reduced output file */
```

```
  /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    /* place header information in the file */
    data_type=2;     /* real_or_complex */
    max_num_alf=nprime*p/2;    /* num_alpha */
    alpha_min=0.0;     /* alpha_min */
    alpha_max=1.0;     /* alpha_max */
    max_num_f=nprime;     /* max_num_f */
    f_min_all= -0.5;     /* f_min_all */
    f_max_all= 0.5;     /* f_max_all */
    fprintf(ofp,"%i %i %e %e %i %e %e\n",data_type,max_num_alf,alpha_min,
     alpha_max,max_num_f,f_min_all,f_max_all);
    for (i=0; i<nprime; i++){  /* group of lines */
      for (j=0; j<(p/2); j++){  /* line in the group  */
      /* place alpha line header information in file */
        curr_alf=(i*p/2)+j;          /* alpha number */
        num_f=nprime-i;              /* num_f  */
        f_min= -0.5+(0.5*i/nprime); /* f_min  */
        f_max=  0.5-(0.5*i/nprime); /* f_max  */
        fprintf(ofp,"%i %i %e %e\n",curr_alf,num_f,f_min,f_max);
        for (k=i; k<nprime; k++){   /* points on the line */
          tempi=(p/2*(nprime-1))-((k-i)*p/2)+j;
          tempj=k;
          numreal=s4[tempi][tempj].r;
          numimag=s4[tempi][tempj].i;
          fprintf(ofp,"%e %e\n",numreal,numimag);
        }  /* end for k=...   */
      }  /* end for j=...   */
    }  /* end for i=...   */
    /*  close the output file  */
    fclose(ofp);
  } /* end else ...   */
} /* end if reduce=0 ... */

else { /* reduce the amount of output  */

  if (((argc==9)&&(argv[7][1]=='a'))||((argc==8)&&(argv[6][1]=='a'))) {
  /* make an ascii output file of the whole spectral plane */
  /* open the output file and place header information in it */
     ofp = fopen(outfile,"w");
     max_num_alf=nprime;
     max_num_f=nprime;
     fprintf(ofp,"%i %i\n",max_num_alf,max_num_f);
     for (i=0; i<max_num_alf; i++){
       for (j=0; j<nprime; j++){
         numreal=s4[i*halfp][j].r;
         numimag=s4[i*halfp][j].i;
         bigmag=sqrt(numreal*numreal + numimag*numimag);
         for (k=1; k<halfp; k++) {  /* look for largest value */
           tempreal=s4[i*halfp+k][j].r;
           tempimag=s4[i*halfp+k][j].i;
           tempmag=sqrt(tempreal*tempreal + tempimag*tempimag);
           if (tempmag>bigmag) {  /* found a larger value */
             bigmag=tempmag;
             numreal=tempreal;
             numimag=tempimag;
```

```
          } /* end if tempmag>bigmag... */
        } /* end if k=1 ... */
        fprintf(ofp,"%e %e\n",numreal,numimag);
      } /* end for j=... */
    } /* end for i=... */
    fclose(ofp);
  } /* end if ascii... */

  else { /* make a plot_sxaf reduced output file */
  /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    /* place header information in the file */
    data_type=2;    /* real_or_complex */
    max_num_alf=nprime;    /* num_alpha */
    alpha_min=0.0;    /* alpha_min */
    alpha_max=1.0;    /* alpha_max */
    max_num_f=nprime;    /* max_num_f */
    f_min_all= -0.5;    /* f_min_all */
    f_max_all= 0.5;    /* f_max_all */
    fprintf(ofp,"%i %i %e %e %i %e %e\n",data_type,max_num_alf,alpha_min,
     alpha_max,max_num_f,f_min_all,f_max_all);
    for (i=0; i<nprime; i++){  /* group of lines */
      /* place alpha line header information in file */
        curr_alf=i;  /* alpha */
        num_f=nprime-i;  /* num_f  */
        f_min= -0.5+(0.5*i/nprime);   /* f_min  */
        f_max= 0.5-(0.5*i/nprime);   /* f_max  */
        fprintf(ofp,"%i %i %e %e\n",curr_alf,num_f,f_min,f_max);
        for (k=i; k<nprime; k++){   /* points on the line */
          tempi=(halfp*(nprime-1))-((k-i)*halfp);
          tempj=k;
          numreal=s4[tempi][tempj].r;
          numimag=s4[tempi][tempj].i;
          bigmag=sqrt(numreal*numreal + numimag*numimag);
          for (j=1; j<halfp; j++){  /* look for the largest value */
            tempreal=s4[tempi+j][tempj].r;
            tempimag=s4[tempi+j][tempj].i;
            tempmag=sqrt(tempreal*tempreal + tempimag*tempimag);
            if (tempmag>bigmag){  /* found a larger value */
              bigmag=tempmag;
              numreal=tempreal;
              numimag=tempimag;
            } /* end if tempmag> ....  */
          } /* end for j=...   */
          fprintf(ofp,"%e %e\n",numreal,numimag);
        } /* end for k=...   */
    } /* end for i=...   */
    /*  close the output file  */
    fclose(ofp);
  } /* end else binary...   */
} /* end  reduce the amount of output */

}  /* end of main */
```

APPENDIX C. SSCA PROGRAM USE

Correct commands are:

for the cross-ssca

```
ssca inputfile1 inputfile2 outputfile n nprime l Oflag
ssca inputfile1 inputfile2 outputfile n nprime l Oflag red
```

or for the auto-ssca

```
ssca inputfile1 outputfile n nprime l Oflag
ssca inputfile1 outputfile n nprime l Oflag red
```

where: inputfile1 is the file containing the x signal samples
inputfile2 is the file containing the y signal samples
outputfile is where the spectrum values will be placed
n is the number of samples to use from the inputfiles
and it must be a power of 2
nprime is the group size of the input datasets and it
must be a power of 2
l is the starting offset of subsequent datasets and
it must be a power of 2
Oflag indicates the output filetype, ascii or binary
-asc for an ascii, MATLAB compatible file,
full cyclic spectral plane
-plo for a *plot_sxaf*, SSPI compatible file,
half cyclic spectral plane
red reduces the amount of data output

note 1: the first two entries in every input file are expected to be the datatype and n. datatype = 1 for real values, 2 for complex values. n is the number of samples contained in the file, one sample per line.

note 2: error may occur using the reduce option if (n-nprime)/nprime is not an integer.

input file format:

```
datatype    n
sample #1
sample #2
   .
   .
   .
sample #n
```

output ASCII MATLAB format:

```
nprime      n
output(1)(1)
output(1)(2)
   .
   .
   .
output(nprime)(n)
```

output SSPI *plot_sxaf* format:

```
datatype (1 for real, 2 for complex)
number of alphasalphaminalphamax
number of freqsfreqminfreqmax

value of alpha #1
number of freqs in #1freqmin_in_#1freqmax_in_#1
output for freqmin_in_#1
   .
   .
   .
output for freqmax_in_#1

      .
      .
      .

value of alpha #alphamax
number of freqs in #alphamaxfreqminfreqmax
output for freqmin_in_#alphamax
   .
   .
   .
output for freqmax_in_#alphamax
```

APPENDIX D. SSCA PROGRAM LISTING

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "/home3/carter/thesis/SSPI/pam/fft.c"
#include "/home3/carter/thesis/SSPI/pam/radix.c"
main(argc,argv)
int argc;
char *argv[];
{
COMPLEX *x,**s3,**dwnconv;
COMPLEX *tempfft,*s1,*y1,**s2,*window;
int i,j,k,type,n,nprime,p,l,r,direction,norm;
int a,cross,file_n,m,reduce,sizes3,redindex;
int tempi,tempj,tempk,templ;
int *outint;
float numreal,numimag,convfac,mainfac;
float bigmag,tempmag,tempreal,tempimag;
float twopi=6.28318530718;
float *outreal;
double z,y;
char *infile1,*infile2,*outfile;
FILE *ifp1,*ifp2,*ofp;
/*  x  passes data to/from fft routine
    s3 holds results of operations after first fft
    s1 holds the x sample values from inputfile 1
    y1 holds the y sample values from inputfile 2
    s2 holds the channelized x input data from s1
                                                    */
/*
   check for the correct number of input arguments
   the correct commands are :
        ssca inputfile outputfile n nprime l Oflag
        ssca inputfile outputfile n nprime l Oflag reduce
        ssca inputfile1 inputfile2 outputfile n nprime l Oflag
        ssca inputfile1 inputfile2 outputfile n nprime l Oflag reduce
   where:  inputfile1 is where the x signal samples are expected to be
           inputfile2 is where the y signal samples are expected to be
           outputfile is where the spectrum values will be put
           n is the number of input samples to use
           nprime is the group size of input datasets
           l is the starting offset of subsequent datasets
           Oflag indicates the output file type
             -asc for an ascii, matlab compatible file
             -plo for an ascii, plot_sxaf compatible file
           reduce is an option to generate a reduced amount of output

   note 1: type and file_n are expected to be on the first line of the inputfile/s
           type = 1 for real, 2 for complex,
           file_n = number of samples following the first line

   note 2: error may occur if (n-nprime) is not evenly divisible by nprime
           when using the reduce option

-------------------------------------------------------------------------------
maintenance history
....written by LCDR Nancy J. Carter, USN    NPGS Monterey CA   01 October 1992
```

```
....15 Oct 92....copied from current version of fam.c to alter to ssca algorithm
             ...modified to perform ssca per Brown/Gardner/Loomis paper
....20 Oct 92....made outputfile ascii format MATLAB compatible
....22 Oct 92....made an output format SSPI plot_sxaf compatible
....24 Nov 92....added option for a reduced amount of output
*/
/*                              */
/*    CHECK INPUT VALUES        */
/*                              */
if ((argc != 7)&&(argc != 8)&&(argc != 9)){
  printf("ssca...fatal error\n");
  printf(".......incorrect number of calling arguments\n");
  printf(".......correct formats are:\n");
  printf(".......  ssca inputfile outputfile n nprime l Oflag\n");
  printf(".......  ssca inputfile outputfile n nprime l Oflag reduce\n");
  printf(".......  ssca inputfile1 inputfile2 n outputfile nprime l Oflag\n");
  printf(".......  ssca inputfile1 inputfile2 n outputfile nprime l Oflag reduce\n");
  printf(".......    where\n");
  printf(".......      inputfile1 contains the x signal samples\n");
  printf(".......      inputfile2 contains the y signal samples\n");
  printf(".......      outputfile will contain the results\n");
  printf(".......      n is the number of input samples to use\n");
  printf(".......      nprime is the group size of input datasets\n");
  printf(".......      l is the offset of subsequent datasets\n");
  printf(".......      Oflag is the output file format\n");
  printf(".......          Oflag = -asc, a MATLAB file is produced\n");
  printf(".......          Oflag = -plo, a plot_sxaf file is produced\n");
  printf(".......      reduce is an option for reduced amount of output\n");
  exit();
 }
if (argc==7) {  /* this is an auto-ssca, no output reduction  */
  cross=0;
  infile1=argv[1];
  outfile=argv[2];
  n=atoi(argv[3]);
  nprime=atoi(argv[4]);
  l=atoi(argv[5]);
  reduce=0;
}
if ((argc==8)&&(argv[7][0]!='r')) {  /* this is a cross-ssca, no data reduction */
  cross=1;
  infile1=argv[1];
  infile2=argv[2];
  outfile=argv[3];
  n=atoi(argv[4]);
  nprime=atoi(argv[5]);
  l=atoi(argv[6]);
  reduce=0;
}
if ((argc==8)&&(argv[7][0]=='r')) {  /* this is an auto-ssca, with data reduction */
  cross=0;
  infile1=argv[1];
  outfile=argv[2];
  n=atoi(argv[3]);
  nprime=atoi(argv[4]);
  l=atoi(argv[5]);
```

```
  reduce=1;
}
if (argc==9) {  /* this is a cross-ssca, with data reduction */
  cross=1;
  infile1=argv[1];
  infile2=argv[2];
  outfile=argv[3];
  n=atoi(argv[4]);
  nprime=atoi(argv[5]);
  l=atoi(argv[6]);
  reduce=1;
}
/* verify that n, nprime and l are powers of 2                    */
i=n%2;
if (i!=0){
  printf("ssca...fatal error\n");
  printf(".......calling argument n is not a power of 2\n");
  exit();
 }
i=nprime%2;
if (i!=0){
  printf("ssca...fatal error\n");
  printf(".......calling argument nprime is not a power of 2\n");
  exit();
 }
j=l%2;
if (j!=0){
  printf("ssca...fatal error\n");
  printf(".......calling argument l is not a power of 2\n");
  exit();
 }
/*                                                                */
/* open input file 1, prepare to get the x signal sample data    */
/*                                                                */
ifp1 = fopen(infile1,"r");
fscanf(ifp1,"%i %i",&type,&file_n);
/* verify that file_n is at least equal to n  */
if (file_n < n){
  printf("ssca...fatal error\n");
  printf(".......inputfile1 does not contain enough samples\n");
  fclose(ifp1);
  exit();
 }
/* find p - the number of datasets(nprime long)  */
p=((n-nprime)/l);
/* find one dimension of final data array */
sizes3=p*l;
/*    */
/*    READ IN DATA SAMPLES   */
/*    */
/* allocate space to read in the x sample values */
/*    */
s1=(COMPLEX*)calloc(n,sizeof(COMPLEX));
if (s1==NULL){
  printf("ssca...insufficient space to allocate s1\n");
  exit();
```

```
 }
if (type==1) {  /* read in the real sample values */
 for (i=0; i<n; i++) {
   fscanf(ifp1,"%e\n",&numreal);
   s1[i].r=numreal;
   s1[i].i=0.0;
 }
}
else {  /* read in the complex sample values */
 for (i=0; i < n; i++){
   fscanf(ifp1,"%e  %e\n",&numreal,&numimag);
   s1[i].r=numreal;
   s1[i].i=numimag;
 }
}
/*    */
/*  close the input file #1   */
/*    */
fclose(ifp1);
/*    */
if (cross==1) {
/*                                                              */
/* open inputfile2, prepare to get the y signal sample data   */
/*                                                              */
  ifp2 = fopen(infile2,"r");
  fscanf(ifp2,"%i %i",&type,&file_n);
/* verify that file_n is at least as big as n    */
  if (file_n < n){
    printf("ssca...fatal error\n");
    printf(".......inputfile2 does not contain enough samples\n");
    fclose(ifp2);
    exit();
  }
/*    */
/* allocate space to read in the y sample values */
/*    */
  y1=(COMPLEX*)calloc(n,sizeof(COMPLEX));
  if (y1==NULL){
    printf("ssca...insufficient space to allocate y1\n");
    exit();
    }
/*                                                              */
/*  read in the complex sample values */
/*                                                              */
  for (i=0; i < n; i++){
    fscanf(ifp2,"%e  %e\n",&numreal,&numimag);
    y1[i].r=numreal;
    y1[i].i=numimag;
  }
/*    */
/*  close the inputfile2   */
/*    */
  fclose(ifp2);
}  /* end if cross = 1  */
/*    */
/*    FORM DATASETS    */
```

```
/*    */
/* allocate space to hold the p-by-nprime datasets  */
/*    */
s2=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (s2==NULL){
  printf("ssca...insufficient space to allocate s2\n");
  exit();
}
for (i=0; i<p; i++){
  s2[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (s2[i]==NULL){
    printf("ssca...insufficient space to allocate s2\n");
    exit();
    }
}
/*                                                              */
/* form p datasets of nprime samples from the original sample stream  */
/*                                                              */
for (i=0; i<p; i++){
  for (j=0; j<nprime; j++){
    k=i*l+j;
    s2[i][j]=s1[k];
  }
}
/*    */
/*    APPLY WINDOW TO DATASETS    */
/*    */
/* allocate space to hold the window multiplicand values */
/*    */
window=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
if (window==NULL){
  printf("ssca...insufficient space to allocate s1\n");
  exit();
}
/*                                          */
/* calculate the window values for the "nprime" sample wide window  */
/*                                          */
for (i=0; i<nprime; i++){
    y=(twopi*i)/(nprime-1);
    z=cos(y);
    window[i].r= 0.54 - (0.46*z);
    window[i].i= window[i].r;
}
/*                                          */
/* apply the window to rows of s2 containing data       */
/*                                          */
for (i=0; i<p; i++){
    for (j=0; j<nprime; j++){
        s2[i][j].i=s2[i][j].i*window[j].i;
        s2[i][j].r=s2[i][j].r*window[j].r;
    }
}
/*    */
/*  FFT EACH DATASET ROW    */
/*    */
/*  allocate space to hold rows of s2 for passing to the FFT routine */
```

```
/*    */
x=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
if (x==NULL){
  printf("ssca...insufficient space to allocate x\n");
  exit();
}
/*    */
/*  allocate space to hold rows of results from the FFT routine */
/*    */
s3=(COMPLEX**)calloc(nprime,sizeof(COMPLEX*));
if (s3==NULL){
  printf("ssca...insufficient space to allocate s3\n");
  exit();
}
for (i=0; i<nprime; i++){
  s3[i]=(COMPLEX*)calloc((p*l),sizeof(COMPLEX));
  if (s3[i]==NULL){
    printf("ssca...insufficient space to allocate s3\n");
    exit();
    }
}
/*                                          */
/* take an FFT of each row of s2            */
/*                                          */
/*    set values in arguments sent to fft  */
direction=1;
norm=1;
for (i=0; i<p; i++){
  /*    copy row of s2 into complex array      */
  for (j=0; j<nprime; j++){
    x[j]=s2[i][j];
    }
  /*    go get FFT performed                   */
  fft(x,nprime,direction,norm);
  /* copy x values into lth column of s3           */
  r=i*l;
  for (j=0; j<nprime; j++){
      s3[j][r]=x[j];
  }
}
/*    */
/*    DOWNCONVERT EACH ROW  */
/*    */
/*  allocate space to hold the downconversion multiplicands  */
/*    */
dwnconv=(COMPLEX**)calloc(p,sizeof(COMPLEX*));
if (dwnconv==NULL){
  printf("ssca...insufficient space to allocate dwnconv\n");
  exit();
}
for (i=0; i<p; i++){
  dwnconv[i]=(COMPLEX*)calloc(nprime,sizeof(COMPLEX));
  if (dwnconv[i]==NULL){
    printf("ssca...insufficient space to allocate dwnconv\n");
    exit();
    }
```

```
/*                                                  */
/*  downconvert each of the transform sequences     */
/*                                                  */
/*      calculate the down conversion multipliers   */
/*                                                  */
mainfac=twopi*l/nprime;
 for (m=0; m<p; m++){
  for (k=0; k<nprime; k++){
    convfac=mainfac*(k-(nprime/2)+1)*m;
    dwnconv[m][k].r=cos(convfac);
    dwnconv[m][k].i= (-1.0)*sin(convfac);
  }
}
/* multiply downconversion factor against each frequency value  */
for (i=0; i<p; i++){
  tempi=i*l;
  for (j=0; j<nprime; j++){
    numreal=s3[j][tempi].r*dwnconv[i][j].r - s3[j][tempi].i*dwnconv[i][j].i;
    s3[j][tempi].i=s3[j][tempi].r*dwnconv[i][j].i + s3[j][tempi].i*dwnconv[i][j].r;
    s3[j][tempi].r=numreal;
  }
}
/*    */
/*    REPLICATE COLUMNS    */
/*    */
for (i=0; i<p; i++) {
  r=i*l;
  /* copy column into l-1 adjacent columns */
  for (j=0; j<nprime; j++) {
    for (k=1; k<l; k++) {
      s3[j][r+k]=s3[j][r];
    }  /* end for k=...  */
  }  /* end for j=...  */
}  /* end for i=... */
/*    */
/*    MULTIPLY COLUMNS    */
/*    */
if (cross==0) {  /* auto-ssca  */
/*    */
/*  multiply columns of s3 with conjugate value of s1  */
/*    */
  for (a=0; a<sizes3; a++) {
    for (j=0; j<nprime; j++){
      s3[j][a].r=(s3[j][a].r*s1[a].r)+(s3[j][a].i*s1[a].i);
      s3[j][a].i=(s3[j][a].i*s1[a].r)-(s3[j][a].r*s1[a].i);
    }
  } /* end for a=0...  */
}  /* end auto-ssca */
else {  /* cross-ssca  */
/*    */
/*  multiply columns of s3 with conjugate value of y1  */
/*    */
  for (a=0; a<sizes3; a++) {
    for (j=0; j<nprime; j++){
      s3[j][a].r=(s3[j][a].r*y1[a].r)+(s3[j][a].i*y1[a].i);
```

```
      s3[j][a].i=(s3[j][a].i*y1[a].r)-(s3[j][a].r*y1[a].i);
    }
  } /* end for a=0...  */
  /*  end cross-ssca  */
/*                                                  */
/*   FFT EACH ROW    */
/*    */
/*    */
/*  allocate space to hold rows of s3 for passing to the FFT routine */
/*    */
tempfft=(COMPLEX*)calloc(sizes3,sizeof(COMPLEX));
if (tempfft==NULL){
  printf("ssca...insufficient space to allocate tempfft  %i by 1\n",sizes3);
  exit();
 }
/*    set values in arguments sent to fft  */
direction=1;
norm=1;
for (j=0; j<nprime; j++){
  /* copy row of s3 into tempfft  */
  for (k=0; k<sizes3; k++){
    tempfft[k]=s3[j][k];
   }
  /*    go get FFT performed                    */
  fft(tempfft,sizes3,direction,norm);
  /* copy tempfft result values back into a row of s3           */
  for (k=0; k<sizes3; k++){
    s3[j][k]=tempfft[k];
   }
} /* end for j=0... */
/*                          */
/*  OUTPUT                  */
/*                          */
if (reduce==0) { /* no output reduction */
  if (((argc==7)&&(argv[6][1]=='a'))||((argc==6)&&(argv[5][1]=='a')))  {
    /* make an ascii output file full spectral plane */
    /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    fprintf(ofp,"%i %i\n",nprime,sizes3);
    /*  write out data values to file for plotting  */
    for (i=0; i<nprime; i++){
      for (j=0; j<sizes3; j++){
        fprintf(ofp,"%e  %e\n",s3[i][j].r,s3[i][j].i);
       }  /* for j=0... */
     }  /* for i=0... */
    /*  close the output file  */
    fclose(ofp);
    } /* if ascii... */
  else {  /* make a plot_sxaf output file half spectral plane */
    /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    a=0;  /* alpha = 0 */
    fprintf(ofp,"%i %i %e %e %i %e %e\n",2,sizes3,0.0,1.0,nprime,-0.5,0.5);
    for (i=0; i<nprime; i++) {  /* group of lines */
      for (j=0; j<(sizes3/nprime); j++) {  /* line in the group  */
        tempreal= -0.5+(0.5*i/nprime);
```

```
        tempimag= 0.5-(0.5*i/nprime);
        fprintf(ofp,"%i %i %e %e\n",a,(nprime-i),tempreal,tempimag);
        a=a+1;
        for (k=0; k<(nprime-i); k++){   /* points on the line  */
          temp1=nprime-k-1;
          tempj=(i+k)*sizes3/nprime+j;
          numreal=s3[temp1][tempj].r;
          numimag=s3[temp1][tempj].i;
          fprintf(ofp,"%e %e\n",numreal,numimag);
          } /* end for k=...  */
        } /* end for j=...  */
      }  /* end for i=...  */
      /*  close the output file  */
      fclose(ofp);
    } /* end if_else */
} /* end if reduce=0... */
if (reduce==1) {  /* reduce the amount of output data */
  if (((argc==8)&&(argv[6][1]=='a'))||((argc==7)&&(argv[5][1]=='a')))  {
    /* make an ascii output file full spectral plane */
    /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    fprintf(ofp,"%i %i\n",nprime,p);
    redindex=1;
    /*  write out data values to file for plotting  */
    for (i=0; i<nprime; i++){
      for (j=0; j<p; j++){
        numreal=s3[i][j*redindex].r;
        numimag=s3[i][j*redindex].i;
        bigmag=sqrt(numreal*numreal + numimag*numimag);
        for (k=1; k<redindex; k++) {  /* look for largest value */
          tempreal=s3[i][j*redindex +k].r;
          tempimag=s3[i][j*redindex +k].i;
          tempmag=sqrt(tempreal*tempreal + tempimag*tempimag);
          if (tempmag>bigmag) {  /* found a bigger value, swap */
            bigmag=tempmag;
            numreal=tempreal;
            numimag=tempimag;
            }  /* end if tempmag>.... */
          } /* end for k=1...  */
        fprintf(ofp,"%e  %e\n",numreal,numimag);
        }  /* for j=0... */
      }  /* for i=0... */
    /*  close the output file  */
    fclose(ofp);
    } /* if ascii... */
  else {  /* make a plot_sxaf output file with reduction half spectral plane */
    /* open the output file and place header information in it */
    ofp = fopen(outfile,"w");
    redindex=sizes3/nprime;
    a=0;  /* alpha = 0 */
    fprintf(ofp,"%i %i %e %e %i %e %e\n",2,nprime,0.0,1.0,nprime,-0.5,0.5);
    for (i=0; i<nprime; i++) {  /* nprime alpha levels */
      tempreal= -0.5+(0.5*i/nprime);
      tempimag= 0.5-(0.5*i/nprime);
      fprintf(ofp,"%i  %i  %e  %e\n",a,(nprime-i),tempreal,tempimag);
      a=a+1;
```

```
      for (k=0; k<(nprime-i); k++){    /* points at the alpha level  */
        tempi=nprime-k-1;
        tempj=(i+k)*redindex;
        numreal=s3[tempi][tempj].r;
        numimag=s3[tempi][tempj].i;
        bigmag=sqrt(numreal*numreal + numimag*numimag);
        for (j=1; j<redindex; j++) {  /* line in the group  */
          tempreal=s3[tempi][tempj +j].r;
          tempimag=s3[tempi][tempj +j].i;
          tempmag=sqrt(tempreal*tempreal + tempimag*tempimag);
          if (tempmag>bigmag) {  /* found a bigger value, swap */
            bigmag=tempmag;
            numreal=tempreal;
            numimag=tempimag;
            }  /* end if tempmag>.... */
          } /* end for j=...  */
          fprintf(ofp,"%e %e\n",numreal,numimag);
        } /* end for k=...  */
      }  /* end for i=...  */
      /*  close the output file  */
      fclose(ofp);
    } /* end if_else */
} /* end if reduce=1... */
}
```

APPENDIX E. SUBFAM PROGRAM USE

Correct command is:

```
subfam n inputfile1 offset1 freqzero1 conj1 inputfile2 offset2
    freqzero2 conj2 outputfile IOflag Itypes
```

where:

n is the number of samples to use from each inputfile, it must be a power of 2

inputfile1 is the first file containing signal samples

offset1 is the initial sample position offset location in inputfile1

freqzero1 is the flag indicating which frequency modification to perform on inputfile1, options are:
- zn - zero neg freqs, shift pos freqs down
- zp - zero pos freqs, shift neg freqs up
- zz - don't do anything to frequency components

conj1 is the flag indicating if conjugation of the data from inputfile1 is to be performed prior to multiplication with data from inputfile2, options:
- y - yes, conjugate data
- n - do not conjugate data

inputfile2 is the second file containing signal samples

offset2 is the initial sample position offset location in inputfile2

freqzero2 is the flag indicating which frequency modification to perform on inputfile2, options are:
- zn - zero neg freqs, shift pos freqs down
- zp - zero pos freqs, shift neg freqs up
- zz - don't do anything to frequency components

conj2 is the flag indicating if conjugation of the data from inputfile2 is to be performed prior to multiplication with data from inputfile1, options:
- y - yes, conjugate data
- n - do not conjugate data

outputfile is where the spectrum values will be placed

IOflag indicates the datatypes of inputfile1, inputfile2 and the outputfile. three characters are given to indicate the datatype of each file. valid datatypes are:
a - file is of type ascii
b - file is of type binary
ex. "abb" means inputfile1 is of type ascii, inputfile2 is of type binary and the outputfile is to be of type binary

Itypes indicates the types of data inside inputfile1 and inputfile2. the outputfile is always complex floating point, real and imaginary components on the same line. two letters must be given to indicate the datatype for each input file. all types are converted internally to complex floating point. valid types are:
r - single real floating point samples per line
i - single integer sample per line
c - two real floating point numbers per line, representing the real and imaginary parts of a single sample
ex. "ic" means inputfile1 has integer samples and inputfile2 has complex floating point samples

example:

```
subfam 8192 data1.dat 64 zn n data2.dat 2156 zp y
    outputA.dat bbb rr
```

input file format:	output file format:
sample #1	magnitude #1 phase #1
sample #2	magnitude #2 phase #2
.	. .
.	. .
.	. .
sample #n	magnitude #n phase #n

APPENDIX F. SUBFAM PROGRAM LISTING

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "/home3/carter/thesis/SSPI/pam/fft.c"
#include "/home3/carter/thesis/SSPI/pam/radix.c"
/* values used in up/down conversion */
#define INTFREQ 49375000.0    /* intermediate signal freq in Hz */
#define SAMPTIME 0.0000051282    /* orig signal sampling period in sec */
main(argc,argv)
int argc;
char *argv[];
{
COMPLEX *s1,*s2,*s3;
int i,n,direction,norm,off1,off2,quarter_n,scaninteger,*readinteger,*z;
float tempreal,tempimag,outputmag,outputphase,numreal,intfreq,t,convfac;
float *readreal,*readimag,*y;
char *infile1,*infile2,*outfile,*conj1,*conj2,*freqzero1,*freqzero2,*ioflag,*itypes;
FILE *ifp1,*ifp2,*ofp;
/*
   subfam is a program which uses signal samples from two input files to calculate
   spectral values in a single diamond of the frequency-alpha plane

   the command is :

        subfam n inputfile1 offset1 freqzero1 conj1 inputfile2 offset2 freqzero2
                conj2 outputfile IOflag Itypes

           n is the number of samples to use from each inputfile, must be pwr of 2

           inputfile1 is the name of the first input file containing signal samples
           offset1 is the initial sample position offset location in inputfile1
           freqzero1 is the flag indicating which frequency modification to perform
             on inputfile1, valid options are:
                 zn - zero negative frequency components, shift positive
                      components down
                 zp - zero positive frequency components, shift negative
                      components up
                 zz - don't do anything to frequency components
           conj1 is the flag indicating if conjugation of the data from inputfile1 is
                 to be performed prior to multiplication with data from inputfile2.
                 valid options are:
                 y - yes conjugate data
                 n - do not conjugate data

           inputfile2 is the name of the second input file containing signal samples
           offset2 is the initial sample position offset location in inputfile2
           freqzero2 is the flag indicating which frequency modification to perform
             on inputfile2, valid options are:
                 zn - zero negative frequency components, shift positive
                      components down
                 zp - zero positive frequency components, shift negative
                      components up
                 zz - don't do anything to frequency components
           conj2 is the flag indicating if conjugation of the data from inputfile2 is
                 to be performed prior to multiplication with data from inputfile1.
                 valid options are:
```

```
                y - yes conjugate data
                n - do not conjugate data

          outputfile is the filename where the spectrum values will be placed

          IOflag indicates the datatypes of inputfile1, inputfile2 and outputfile.
               three letters must be given to indicate the datatypes for all files.
               valid datatypes are:
               a - file is of type ascii
               b - file is of type binary
               ex. "abb" means inputfile1 is of type ascii,
                         inputfile2 is of type binary, and
                         the outputfile is to be of type binary

          Itypes indicates the number types of data inside inputfile1 and
               inputfile2.  the outputfile is always complex floating point,
               real and imaginary components on the same line.  two letters
               must be given to indicate the number type for each input file.
               all number types are converted to complex floating point.
               valid number types are:
               r - single real floating point sample per line
               i - single integer sample per line
               c - two floating point numbers per sample on each line
               ex. "ic" means inputfile1 has integer samples and
                        inputfile2 has complex floating point samples

   sample useage:

     subfam 8192 data1.dat 64 zn n data2.dat 2156 zp y outputA.dat bbb rr

   maintenance history
... created 01 October 1992, LCDR Nancy J. Carter, USN at NPGS Monterey CA
... 10.15.92 - added freqzero options - njc
... 10.16.92 - added error check on conj1 and conj2 - njc
... 10.25.92 - changed down/up shifting to same code as in old version

------------------------------------------------------------------------  */
/* check for the correct number of input arguments                        */
if (argc != 13){
  printf("subfam...fatal error...incorrect number of calling arguments\n");
  printf("\n");
  printf("....correct format is:  \n");
  printf("  subfam n inputfile1 offset1 freqzero1 conj1 inputfile2 offset2 \n");
  printf("        freqzero2 conj2 outputfile IOflag Itypes\n");
  printf("\n");
  printf("\n");
  printf("......n is the number of samples to use from each inputfile, must be pwr of 2\n"
  printf("......inputfile1 is a file containing the signal samples\n");
  printf("......offset1 is the initial sample position offset in inputfile1\n");
  printf("......freqzero1 indicates type of freq zeroing and shifting to do on inputfile1\
  printf("          zn....zero negative freqs and shift down\n");
  printf("          zp....zero positive freqs and shift up\n");
  printf("          zz....do not zero anything, do not shift\n");
  printf("......conj1 indicates if conjugation of data from inputfile1 is\n");
  printf("          desired before multiplication\n");
  printf("          y....yes conjugate data\n");
```

```
  printf("          n....no do not conjugate data\n");
  printf("          usual setup is conj1 = n, conj2 = y\n");
  printf("......inputfile2 is a file containing the signal samples\n");
  printf("......offset2 is the initial sample position offset in inputfile2\n");
  printf("......freqzero2 indicates type of freq zeroing and shifting to do on inputfile2
  printf("          zn....zero negative freqs and shift down\n");
  printf("          zp....zero positive freqs and shift up\n");
  printf("          zz....do not zero anything, do not shift\n");
  printf("......conj2 indicates if conjugation of data from inputfile2 is\n");
  printf("          desired before multiplication\n");
  printf("          y....yes conjugate data\n");
  printf("          n....no do not conjugate data\n");
  printf("          usual setup is conj1 = n, conj2 = y\n");
  printf("......outputfile is where the spectrum values will be placed\n");
  printf("......IOflag indicates datatypes of inputfile1,inputfile2 and outputfile\n");
  printf("          aaa..............ascii,ascii,ascii\n");
  printf("          aab..............ascii,ascii,binary\n");
  printf("          aba..............ascii,binary,ascii\n");
  printf("          abb..............ascii,binary,binary\n");
  printf("          baa..............ascii,ascii,ascii\n");
  printf("          bab..............ascii,ascii,binary\n");
  printf("          bba..............binary,binary,ascii\n");
  printf("          bbb..............binary,binary,binary\n");
  printf("......Itypes indicates number types of input samples\n");
  printf("          r................single real float\n");
  printf("          i................single integer\n");
  printf("          c................complex, two floats\n");
  printf("\n");
  printf("....sample useage:\n");
  printf(" subfam 8192 data1.dat 64 zn n data2.dat 2156 zp y outputA.dat bbb rr\n");
  exit();
  }
n=atoi(argv[1]);
infile1=argv[2];
off1=atoi(argv[3]);
freqzero1=argv[4];
conj1=argv[5];
infile2=argv[6];
off2=atoi(argv[7]);
freqzero2=argv[8];
conj2=argv[9];
outfile=argv[10];
ioflag=argv[11];
itypes=argv[12];

/* verify that n is a power of 2                */
i=n%2;
if (i!=0){
  printf("subfam...fatal error\n");
  printf(".......calling argument n is not a power of 2\n");
  exit();
  }
/* verify that conj1 & conj2 are y or n  */
if (((conj1[0]!='y')&&(conj1[0]!='n'))||((conj2[0]!='y')&&(conj2[0]!='n'))) {
  printf("subfam...fatal error\n");
  printf(".......conj1 and conj2 must be set to y or n\n");
```

```
  exit();
  }
/* verify that itypes are r, i or c    */
if (((itypes[0]!='r')&&(itypes[0]!='i')&&(itypes[0]!='c'))||
    ((itypes[1]!='r')&&(itypes[1]!='i')&&(itypes[1]!='c'))) {
  printf("subfam...fatal error\n");
  printf(".......Itypes must be set to r, i or c\n");
  exit();
  }
/*          */
/* allocate space for the 1-D data arrays  */
/*          */
s1=(COMPLEX*)calloc(n,sizeof(COMPLEX));
if (s1==NULL){
  printf("subfam...insufficient space to allocate s1\n");
  exit();
  }
s2=(COMPLEX*)calloc(n,sizeof(COMPLEX));
if (s2==NULL){
  printf("subfam...insufficient space to allocate s2\n");
  exit();
  }
/* allocate space to read in binary data                    */
y=(float*)malloc(1*sizeof(float));
z=(int*)malloc(1*sizeof(int));
/*                                                          */
/* INPUT */
/*                                                          */
/* open input files, get the signal samples   */
/*                                                          */
switch (ioflag[0]) {
   case 'a':    /* inputfile1 is an ASCII file */
     if (( ifp1 = fopen(infile1,"r")) == NULL )  {
       printf("subfam...unable to open ascii inputfile1\n");
       exit();
       }
     /*  move to the desired starting offset position in the file  */
     for (i=0;i<off1;i++){
       switch (itypes[0]) {
         case 'r':  /* single real float per line */
           if (fscanf(ifp1,"%e",&tempreal)!=1){
             printf("subfam...EOF encountered while attempting to reach offset1\n");
             fclose(ifp1);
             exit();
             }
             break;
         case 'i':  /* single integer per line */
           if (fscanf(ifp1,"%i",scaninteger)!=1){
             printf("subfam...EOF encountered while attempting to reach offset1\n");
             fclose(ifp1);
             exit();
             }
             break;
         default:  /* complex, two floats per line */
           if (fscanf(ifp1,"%e %e",&tempreal,&tempimag)!=2){
             printf("subfam...EOF encountered while attempting to reach offset1\n");
```

```
            fclose(ifp1);
            exit();
            }
        }  /* end switch itypes[0] */
      } /* end if i=0;i<off1  */

    /* read in n data samples */
    for (i=0;i<n;i++){
      switch (itypes[0]) {
        case 'r':  /* single real float per line */
          if (fscanf(ifp1,"%e",&s1[i].r)!=1){
            printf("subfam...EOF reached while reading inputfile1\n");
            fclose(ifp1);
            exit();
            }
          break;
        case 'i':  /* single integer per line */
          if (fscanf(ifp1,"%i",scaninteger)!=1){
            printf("subfam...EOF reached while reading inputfile1\n");
            fclose(ifp1);
            exit();
            }
          s1[i].r=scaninteger;
        break;
        default:  /* complex, two floats per line */
          if (fscanf(ifp1,"%e %e",&s1[i].r,&s1[i].i)!=2){
            printf("subfam...EOF reached while reading inputfile1\n");
            fclose(ifp1);
            exit();
            }
        }  /* end switch itypes[0] */
      }  /* end for i=0 i<n  */
    fclose(ifp1);
    break;

  case 'b':    /* inputfile1 is a BINARY file */
    if (( ifp1 = fopen(infile1,"rb")) == NULL )  {
      printf("subfam...unable to open binary inputfile1\n");
      exit();
      }
    /*  move to the desired starting offset position in the file  */
    for (i=0;i<off1;i++){
      switch (itypes[0]) {
        case 'r':  /* single real float per line */
          if (fread(y,sizeof(*y),1,ifp1)!=1){
            printf("subfam...EOF encountered while attempting to reach offset1\n");
            fclose(ifp1);
            exit();
            }
          break;
        case 'i':  /* single integer per line */
          if (fread(z,sizeof(*z),1,ifp1)!=1){
            printf("subfam...EOF encountered while attempting to reach offset1\n");
            fclose(ifp1);
            exit();
            }
```

```
            break;
          default:  /* complex, two floats per line */
            if (fread(y,sizeof(*y),1,ifp1)!=1){
              printf("subfam...EOF encountered while attempting to reach offset1\n");
              fclose(ifp1);
              exit();
              }
            if (fread(y,sizeof(*y),1,ifp1)!=1){
              printf("subfam...EOF encountered while attempting to reach offset1\n");
              fclose(ifp1);
              exit();
              }
          }  /* end switch itypes[0] */
        }  /* end if i=0;i<off1    */

      /* read in n data samples */
      for (i=0;i<n;i++){
        switch (itypes[0]) {
          case 'r':  /* single real float per line */
            if (fread(y,sizeof(*y),1,ifp1)!=1){
              printf("subfam...EOF reached while reading inputfile1\n");
              fclose(ifp1);
              exit();
              }
            s1[i].r=*y;
            break;
          case 'i':  /* single integer per line */
            if (fread(z,sizeof(*z),1,ifp1)!=1){
              printf("subfam...EOF reached while reading inputfile1\n");
              fclose(ifp1);
              exit();
              }
              s1[i].r=*z;
          break;
          default:  /* complex, two floats per line */
            if (fread(y,sizeof(*y),1,ifp1)!=1){
              printf("subfam...EOF reached while reading inputfile1\n");
              fclose(ifp1);
              exit();
              }
            s1[i].r=*y;
            if (fread(y,sizeof(*y),1,ifp1)!=1){
              printf("subfam...EOF reached while reading inputfile1\n");
              fclose(ifp1);
              exit();
              }
            s1[i].r=*y;
          }  /* end switch itypes[0] */
        }  /* end for i=0 i<n  */
      fclose(ifp1);
      break;
    default:     /* invalid format specified for inputfile1 */
      printf("subfam...invalid format specified for inputfile1\n");
      exit();
    } /* end switch ioflag[0] */
```

```
switch (ioflag[1]) {
   case 'a':    /* inputfile2 is an ASCII file */
     if (( ifp2 = fopen(infile2,"r")) == NULL )  {
       printf("subfam...unable to open ascii inputfile2\n");
       exit();
       }
     /*  move to the desired starting offset position in the file  */
     for (i=0;i<off2;i++){
       switch (itypes[1]) {
         case 'r':  /* single real float per line */
           if (fscanf(ifp2,"%e",&tempreal)!=1){
             printf("subfam...EOF encountered while attempting to reach offset2\n");
             fclose(ifp2);
             exit();
             }
             break;
         case 'i':  /* single integer per line */
           if (fscanf(ifp2,"%i",scaninteger)!=1){
             printf("subfam...EOF encountered while attempting to reach offset2\n");
             fclose(ifp2);
             exit();
             }
             break;
         default:  /* complex, two floats per line */
           if (fscanf(ifp2,"%e %e",&tempreal,&tempimag)!=2){
             printf("subfam...EOF encountered while attempting to reach offset2\n");
             fclose(ifp2);
             exit();
             }
         }  /* end switch itypes[1] */
       }  /* end if i=0;i<off2  */

     /* read in n data samples */
     for (i=0;i<n;i++){
       switch (itypes[1]) {
         case 'r':  /* single real float per line */
           if (fscanf(ifp2,"%e",&s2[i].r)!=1){
             printf("subfam...EOF reached while reading inputfile2\n");
             fclose(ifp2);
             exit();
             }
           break;
         case 'i':  /* single integer per line */
           if (fscanf(ifp2,"%i",scaninteger)!=1){
             printf("subfam...EOF reached while reading inputfile2\n");
             fclose(ifp2);
             exit();
             }
             s2[i].r=scaninteger;
         break;
         default:  /* complex, two floats per line */
           if (fscanf(ifp2,"%e %e",&s2[i].r,&s2[i].i)!=2){
             printf("subfam...EOF reached while reading inputfile2\n");
             fclose(ifp2);
             exit();
             }
```

```
        }  /* end switch itypes[1] */
      }  /* end for i=0 i<n  */
    fclose(ifp2);
    break;

   case 'b':     /* inputfile2 is a BINARY file */
    if (( ifp2 = fopen(infile2,"rb")) == NULL )  {
      printf("subfam...unable to open binary inputfile2\n");
      exit();
      }
    /*  move to the desired starting offset position in the file  */
    for (i=0;i<off2;i++){
      switch (itypes[1]) {
        case 'r':  /* single real float per line */
          if (fread(readreal,sizeof(*readreal),1,ifp2)!=1){
            printf("subfam...EOF encountered while attempting to reach offset2\n");
            fclose(ifp2);
            exit();
            }
            break;
        case 'i':  /* single integer per line */
          if (fread(readinteger,sizeof(*readinteger),1,ifp2)!=1){
            printf("subfam...EOF encountered while attempting to reach offset2\n");
            fclose(ifp2);
            exit();
            }
            break;
        default:  /* complex, two floats per line */
          if (fread(readreal,sizeof(*readreal),1,ifp2)!=1){
            printf("subfam...EOF encountered while attempting to reach offset2\n");
            fclose(ifp2);
            exit();
            }
          if (fread(readimag,sizeof(*readimag),1,ifp2)!=1){
            printf("subfam...EOF encountered while attempting to reach offset2\n");
            fclose(ifp2);
            exit();
            }
        }  /* end switch itypes[1] */
      }  /* end if i=0;i<off2    */

    /* read in n data samples */
    for (i=0;i<n;i++){
      switch (itypes[1]) {
        case 'r':  /* single real float per line */
          if (fread(readreal,sizeof(*readreal),1,ifp2)!=1){
            printf("subfam...EOF reached while reading inputfile2\n");
            fclose(ifp2);
            exit();
            }
          s2[i].r=*readreal;
          break;
        case 'i':  /* single integer per line */
          if (fread(readinteger,sizeof(*readinteger),1,ifp2)!=1){
            printf("subfam...EOF reached while reading inputfile2\n");
            fclose(ifp2);
```

```
            exit();
            }
            scanInteger=*readInteger;
            s2[i].r=scanInteger;
          break;
          default:  /* complex, two floats per line */
            if (fread(readReal,sizeof(*readReal),1,ifp2)!=1){
              printf("subfam...EOF reached while reading inputfile2\n");
              fclose(ifp2);
              exit();
              }
            s2[i].r=*readReal;
            if (fread(readImag,sizeof(*readImag),1,ifp2)!=1){
              printf("subfam...EOF reached while reading inputfile2\n");
              fclose(ifp2);
              exit();
              }
            s2[i].i=*readImag;
          }  /* end switch iTypes[1] */
        }  /* end for i=0 i<n  */
      fclose(ifp2);
       break;
     default:      /* invalid format specified for inputfile2 */
      printf("subfam...invalid format specified for inputfile2\n");
      exit();
   } /* end switch ioflag[1]  */
/*                            */
/* ZEROIZE REQUESTED FREQUENCY SIDE AND SHIFT APPROPRIATELY  */
/*                            */
quarter_n=n/4;
switch (freqzero1[1]) {
  case 'n':    /* zero negative and shift down */
    direction=1;  /* request a forward transform time-to-freq */
    norm=0;
    fft(s1,n,direction,norm);  /* results returned in s1  */
    for (i=0; i<(n/2); i++){ /* zero lower half */
      s1[i].r=0;
      s1[i].i=0;
      }
    direction=-1;  /* request an inverse transform freq-to-time  */
    norm=2;
    fft(s1,n,direction,norm); /* results returned in s1 */
/* downconvert */
    t=0.0;               /* starting time          */
    for (i=0; i<n; i++) {
      convfac=TWOPI*INTFREQ*t;
      tempreal=cos(convfac);
      tempImag=(-1)*sin(convfac);
      numreal=s1[i].r*tempreal - s1[i].i*tempImag;
      s1[i].i=s1[i].r*tempImag+s1[i].i*tempreal;
      s1[i].r=numreal;
      t=t+SAMPTIME;
    }
    break;
  case 'p':    /* zero positive and shift up   */
    direction=1;  /* request a forward transform time-to-freq */
```

```
    norm=0;
    fft(s1,n,direction,norm);  /* results returned in s1  */
    for (i=(n/2); i<n; i++){ /* zero upper half */
      s1[i].r=0;
      s1[i].i=0;
      }
    direction=-1;  /* request an inverse transform freq-to-time  */
    norm=2;
    fft(s1,n,direction,norm); /* results returned in s1 */
/* upconvert */
    t=0.0;              /* starting time          */
    for (i=0; i<n; i++) {
      convfac=TWOPI*INTFREQ*t;
      tempreal=cos(convfac);
      tempimag=sin(convfac);
      numreal=s1[i].r*tempreal - s1[i].i*tempimag;
      s1[i].i=s1[i].r*tempimag+s1[i].i*tempreal;
      s1[i].r=numreal;
      t=t+SAMPTIME;
    }
    break;
  case 'z':    /* do nothing */
    break;
  default:     /* invalid format specified for freqzero1 */
    printf("subfam...invalid format specified for freqzero1\n");
    exit();
  } /* end switch */
switch (freqzero2[1]) {
  case 'n':    /* zero negative and shift down */
    direction=1;  /* request a forward transform time-to-freq */
    norm=0;
    fft(s2,n,direction,norm);  /* results returned in s2  */
    for (i=0; i<(n/2); i++){  /* zero lower half */
      s2[i].r=0;
      s2[i].i=0;
      }
    direction=-1;  /* request an inverse transform freq-to-time  */
    norm=2;
    fft(s2,n,direction,norm); /* results returned in s2 */
/* downconvert */
    t=0.0;              /* starting time          */
    for (i=0; i<n; i++) {
      convfac=TWOPI*INTFREQ*t;
      tempreal=cos(convfac);
      tempimag=(-1)*sin(convfac);
      numreal=s2[i].r*tempreal - s2[i].i*tempimag;
      s2[i].i=s2[i].r*tempimag+s2[i].i*tempreal;
      s2[i].r=numreal;
      t=t+SAMPTIME;
    }
    break;
  case 'p':    /* zero positive and shift up   */
    direction=1;  /* request a forward transform time-to-freq */
    norm=0;
    fft(s2,n,direction,norm);  /* results returned in s2  */
    for (i=(n/2); i<n; i++){ /* zero upper half */
```

```
      s2[i].r=0;
      s2[i].i=0;
      }
    direction=-1;  /* request an inverse transform freq-to-time  */
    norm=2;
    fft(s2,n,direction,norm); /* results returned in s2 */
/* upconvert */
    t=0.0;                  /* starting time          */
    for (i=0; i<n; i++) {
      convfac=TWOPI*INTFREQ*t;
      tempreal=cos(convfac);
      tempimag=sin(convfac);
      numreal=s2[i].r*tempreal - s2[i].i*tempimag;
      s2[i].i=s2[i].r*tempimag+s2[i].i*tempreal;
      s2[i].r=numreal;
      t=t+SAMPTIME;
    }
    break;
  case 'z':    /* do nothing */
    break;
  default:     /* invalid format specified for freqzero2 */
    printf("subfam...invalid format specified for freqzero2\n");
    exit();
  } /* end switch */
/*        */
/* MULTIPLICATION  */
/*        */
/* create space for the output array   */
/*        */
s3=(COMPLEX*)calloc(n,sizeof(COMPLEX));
if (s3==NULL){
  printf("subfam...insufficient space to allocate s3\n");
  exit();
  }
if ((conj1[0]=='y')&&(conj2[0]=='y')) { /* multiply conjugate s1 times conjugate s2 */
  for (i=0;i<n;i++){
    s3[i].r=s1[i].r*s2[i].r-s1[i].i*s2[i].i;
    s3[i].i=(-1)*(s1[i].i*s2[i].i+s1[i].i*s2[i].r);
    }
  }
if ((conj1[0]=='y')&&(conj2[0]=='n')) { /* multiply conjugate s1 times s2 */
  for (i=0;i<n;i++){
    s3[i].r=s1[i].r*s2[i].r+s1[i].i*s2[i].i;
    s3[i].i=s1[i].r*s2[i].i-s1[i].i*s2[i].r;
    }
  }
if ((conj1[0]=='n')&&(conj2[0]=='y')) { /* multiply s1 times conjugate s2 */
  for (i=0;i<n;i++){
    s3[i].r=s1[i].r*s2[i].r+s1[i].i*s2[i].i;
    s3[i].i=s1[i].i*s2[i].r-s1[i].r*s2[i].i;
    }
  }
if ((conj1[0]=='n')&&(conj2[0]=='n')) { /* multiply s1 times s2 */
  for (i=0;i<n;i++){
    s3[i].r=s1[i].r*s2[i].r-s1[i].i*s2[i].i;
    s3[i].i=s1[i].r*s2[i].i+s1[i].i*s2[i].r;
```

```
    }
  }
/* take an FFT of n-point s3   */
/*                                        */
/*    set values in arguments sent to fft  */
direction=1;  /* request a forward transform time-to-freq */
norm=0;
fft(s3,n,direction,norm);  /* results returned in s3  */
/*          */
/* OUTPUT */
/*          */
/* take magnitude and phase of s3 and place in the output file */
/*          */
switch (ioflag[2]) {
   case 'a':    /* outputfile is an ascii file */
     if (( ofp = fopen(outfile,"w")) == NULL )  {
       printf("subfam...unable to open ascii outputfile\n");
       exit();
       }
     /*  write out data to the ascii file             */
      for (i=0;i<n;i++){
        outputmag=sqrt((s3[i].r*s3[i].r)+(s3[i].i*s3[i].i));
        outputphase=atan(s3[i].i/s3[i].r);
        fprintf(ofp,"%-16.6e %-16.6e\n",outputmag,outputphase);
        }
      /*  close the output file  */
      fclose(ofp);
      break;
    case 'b':     /* outputfile is a binary file */
     if (( ofp = fopen(outfile,"wb")) == NULL )  {
       printf("subfam...unable to open binary outputfile\n");
       exit();
       }
     /*  write out data to the binary file             */
     for (i=0;i<n;i++){
        outputmag=sqrt((s3[i].r*s3[i].r)+(s3[i].i*s3[i].i));
        outputphase=atan(s3[i].i/s3[i].r);
        *y=outputmag;
       if (fwrite(y,sizeof(*y),1,ofp)!=1){
         printf("subfam...error while writing outputfile\n");
         fclose(ofp);
         exit();
         }
        *y=outputphase;
       if (fwrite(y,sizeof(*y),1,ofp)!=1){
         printf("subfam...error while writing outputfile\n");
         fclose(ofp);
         exit();
         }
      }  /*  end if i=0 ...  */
      fclose(ofp);
      break;
    default:      /* invalid format specified for outputfile */
     printf("subfam...invalid format specified for outputfile\n");
     exit();
    }  /* end switch ioflag[2]  */
```

```
} /* end of main */
```

LIST OF ABBREVIATIONS

BPSK	binary phase shift keying
FAM	FFT accumulation method
FFT	fast fourier transform
FSM	frequency smoothing method
PAM	pulse amplitude modulation
QAM	quadrature amplitude modulation
QPSK	quadrature phase shift keying
SCF	spectral correlation function
SSCA	strip spectral correlation algorithm
SSPI	Statistical Signal Processing, Inc.
SUBFAM	sub-FFT accumulation method

LIST OF REFERENCES

1. Gardner, W.A., *Statistical Spectral Analysis*, Prentice Hall, 1988.

2. Gardner, W.A., "Exploitation of Spectral Redundancy In Cyclostationary Signals," *IEEE Signal Processing Magazine*, April 1991.

3. National Security Agency, *Spread Spectrum Signals and Techniques Handbook*, Third Edition, Radian Corporation, 1981.

4. Schell, S.V., and Spooner, C.M., *Cyclic Spectral Analysis Software Package*, Version 2.0, Statistical Signal Processing, Inc., Yountville CA, 1991.

5. Roberts, R.S., Brown, W.A. and Loomis, H.H., "Computationally Efficient Algorithms for Cyclic Spectral Analysis," *IEEE Signal Processing Magazine*, April 1991.

6. PRO-MATLAB™ for Sun Workstations, The MathWorks, Inc., Natick MA, January 31, 1990.

7. Loomis, H.H., Notes for an algorithm to compute cycle frequencies for a single spectral frequency in the alpha-frequency plane, Naval Postgraduate School, 1991 (unpublished).

8. Strum, R.D., and Kirk, D.E., *First Principles of Discrete Systems and Digital Signal Processing*, Addison-Wesley, 1988.

9. Brown, R.A. III, and Loomis, H.H. Jr., "Digital Implementations of Spectral Correlation Analyzers," *IEEE Signal Processing Magazine*, (to be published).

10. Roberts, R.S., and Loomis, H.H. Jr., "Parallel Computation Structures for a Class of Cyclic Spectral Analysis Algorithms," *IEEE Signal Processing Magazine*, February 3, 1992.

11. Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, 1990.

12. Schalkoff, R.J., *Digital Image Processing and Computer Vision*, John Wiley & Sons, Inc., 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Prof. H.H. Loomis Jr., Code EC/Lm Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
5.	R.F. Bernstein Jr., Code EC/Be Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6.	LCDR Nancy J. Carter, USN Naval Security Group Support Activity 3801 Nebraska Ave. N.W. Washington, DC 20393-5220	1
7.	NRL Attn: LCDR Oxborough, Code 9110 4555 Overlook Ave., S.W. Washington, DC 20375-5000	1
8.	SSPI Attn: Dr. W.A. Gardner 6950 Yount St. Yountville, CA 94599	1
9.	Signal Science, Inc. Attn: Ms. Carolyn Koenig 2985 Kifer Rd. Santa Clara, CA 95051	1